НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

«На правах рукопису»
УДК 004.056

«До захисту допущено»
В.о. завідувача кафедри
_____ М.В.Грайворонський
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності: 125 Кібербезпека

на тему: Удосконалення механізму перевірки сертифікатів застосунків MacOS

Виконав (-ла): студент (-ка) 2 курсу, групи ФБ-71мп
(шифр групи)

Тіхоничев Роман Миколайович
(прізвище, ім’я, по батькові)

Науковий керівник к.т.н., доц. Стьопочкіна Ірина Валеріївна_ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент к.т.н., доцент ФІОТ Жданова О.Г. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ
Кафедра інформаційної безпеки

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою
Спеціальність (спеціалізація) – 125 Кібербезпека («Системи і технології кібербезпеки)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ М.В.Грайворонський
(підпис)

«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Тіхоничеву Роману Миколайовичу

1. Тема дисертації: Удосконалення механізму перевірки сертифікатів застосунків MacOS

науковий керівник дисертації к.т.н., доц. Стьопочкіна Ірина Валеріївна,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «15» листопада 2018 р. № 4171-с

2. Термін подання студентом дисертації 12.12.2018 р.

3. Об'єкт дослідження _____
_____

4. Вихідні дані _____
_____
_____
_____

5. Перелік завдань, які потрібно розробити _____
_____
_____
_____
_____
_____

6. Орієнтовний перелік ілюстративного матеріалу _____
_____
_____

7. Орієнтовний перелік публікацій _____

_____

_____

8. Консультанти розділів дисертації*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка

Студент _____ (підпис) _____ (ініціали, прізвище)

Науковий керівник дисертації _____ (підпис) _____ (ініціали, прізвище)

* Консультантом не може бути зазначено наукового керівника магістерської дисертації.

РЕФЕРАТ

Робота обсягом 95 сторінок містить 47 ілюстрації, 23 таблиць та 16 літературних посилань.

Метою даної кваліфікаційної роботи є удосконалення існуючого механізму перевірки сертифікатів застосунків MacOS.

Об’єктом дослідження є механізми перевірки сертифікатів застосунків MacOS.

Предметом дослідження є вразливість механізму сертифікації застосунків під MacOS та способи її усунення.

Під час роботи визначені основні системи перевірки сертифікатів MacOS, детально досліджено принцип їх роботи, проаналізовано вразливість даного механізму, та розроблено метод який запобігає її використання.

Результати роботи можуть бути використані при побудові систем захисту MacOS.

ПІДПИС ВИКОНУВАНОГО КОДУ, МЕХАНІЗИ ПЕРЕВІРКИ СЕРТИФІКАТІВ, FAT-BINARIES, ВДОСКОНАЛЕННЯ МЕХАНІЗМУ ПЕРЕВІРКИ СЕРТИФІКАТІВ, GATEKEEPER, CODESIGN. APPLICATION SANDBOX.

РЕФЕРАТ

Работа объемом 95 страниц содержит 47 иллюстрации, 23 таблиц и 16 литературных ссылок.

Целью данной квалификационной работы является совершенствование существующего механизма проверки сертификатов приложений MacOS.

Объектом исследования являются механизмы проверки сертификатов приложений MacOS.

Предметом исследования является уязвимость механизма сертификации приложений под MacOS и способы ее устранения.

Во время работы определены основные системы проверки сертификатов MacOS, подробно исследованы принцип их работы, проанализированы уязвимость данного механизма, и разработан метод который предотвращает ее использования.

Результаты работы могут быть использованы при построении систем защиты MacOS.

ПОДПИСЬ ИСПОЛНЯЕМОГО КОДА, МЕХАНИЗМ ПРОВЕРКИ СЕРТИФИКАТО, FAT-BINARIES, СОВЕРШЕНСТВОВАНИЕ МЕХАНИЗМА ПРОВЕРКИ СЕРТИФИКАТОВ, GATEKEEPER, CODESIGN. APPLICATION SANDBOX.

ABSTRACT

The work volume 95 pages contains 47 illustrations, 23 tables and 16 literary references.

The purpose of this qualification work is to improve the mechanism for MacOS applications certificates verification

The object of the study is the mechanisms for MacOS applications certificates verification.

The subject of the study is the vulnerability of mechanism for MacOS applications certificates verification, and how to fix it.

During the work the main systems for verifying the certificates of MacOS were determined, the principle of their work was analyzed in detail, the vulnerability of this mechanism was analyzed, and a method was developed that prevents use of this vulnerability.

Work results can be used to build new MacOS protection systems.

CODE SIGNATURE, MECHANIZATIONS OF CERTIFICATES VERIFICATION, FAT-BINARIES,IMPROVEMENT THE MECHANISM FOR MAC OS APPLICATIONS CERTIFICATE VERIFICATION, GATEKEEPER, CODESIGN. APPLICATION SANDBOX.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНИХ ТЕРМІНІВ

Public key infrastructure (PKI) - інтегрований комплекс методів та засобів (набір служб), призначених забезпечити впровадження та експлуатацію криптографічних систем із відкритими ключами. [1]

Codesign - підпис виконуваного коду.

Developer Tools Access (DTA) - набір інструментів які використовуються для розробки та аналізу застосунків.

Fat-binaries (Universal) - Виконувані файли які містять в собі скомпільований код для різних архітектур.

API (англ.Application Programming Interface) – опис процедур, структур даних, методів за допомогою яких одна програма може взаємодіяти з іншою.

Apple - американська технологічна компанія з офісом в силіконовій долині яка проектує та розробляє побутову електроніку, програмне забезпечення та онлайн-сервіси. [2]

PowerPC - мікропроцесорна RISC-архітектура, створена в 1991 році альянсом компаній Apple-IBM-Motorola

WWDC - щорічна конференція розробників для платформи Macintosh. Проводиться компанією Apple Computer в Сан Франциско, Каліфорнія. [3]

Kernel Extensions (Kext) - це розширення ядра операційної системи MacOS.

Apple Developer ID - система аутентифікації, яка використовується в багатьох продуктах Apple, зокрема в iTunes Store, App Store, iCloud тощо. [4]

ОС - скорочення від операційна система

App Store - розділ онлайн супермаркету iTunes Store, що продає власникам мобільних телефонів iPhone та Mac різні застосунки.

Safari.app - браузер, розроблений корпорацією Apple і входить до складу операційних систем Mac OS X і iOS.

ВСТУП

Актуальність дослідження зумовлена важливістю цифрового підпису застосунків, який широко використовується для підтвердження цілісності застосунка та його авторства. Цей спосіб дає змогу запобігти несанкціонованному впровадженню програмних закладень; він рекомендований для всіляких застосунків і бібліотек; при відсутності цифрового підпису застосунка чи бібліотеки, які викликаються в програмному коді; автоматичні аналізатори кода сповіщають про це як про потенційну вразливість. Ці факти свідчать про те, що механізм перевірки сертифіката застосунку є необхідною складовою засобів захисту, в тому числі і в MacOS. З іншого боку, нещодавно знайдені в цьому механізмі недоліки призводять до неправильної перевірки сертифікатів.

Тому метою роботи є: Дослідження причин слабкості механізма перевірки сертифікатів застосунків в MacOS; та виявлення присутності вразливості для довільних випадків.

Задачі дослідження:

проаналізувати механізми сертифікації, які використовуються в програмних продуктах MacOS;

виявити схему та послідовність взаємодій, які використовуються під час сертифікації застосунків MacOS;

проаналізувати будову Mach-O Fat бінарних файлів;

проаналізувати наявну вразливість системи перевірки сертифікатів Fat бінарних файлів;

скласти модель загроз, потенційно спричинених вразливістю;

запропонувати методику перевірки наявності вразливості для довільного застосунка MacOS;

розробити програмне рішення, яке втілює основні положення запропонованої методики.

Об’єкт дослідження: Механізми сертифікації застосунків

Предмет дослідження: вразливості механізму сертифікації застосунків під MacOS

Практичне значення роботи: результати роботи можуть бути використані як користувачами MacOs для упевненості в надійності механізмів перевірки сертифікатів; так і розробниками програмного забезпечення під MacOS.

1. ЗАГАЛЬНІ ПРИНЦИПИ РОБОТИ МЕХАНІЗМІВ СЕРТИФІКАЦІЇ

1.1 Що таке підпис виконуваного коду

Підпис виконуваного коду - це процес, при якому в безпосередньо виконувані файли або скрипти вбудовуються додатково електронні підписи, що дозволяють провести перевірку їх авторства або цілісність, часто з використанням геш-сум.

Підпис виконуваного коду допомагає вирішити відразу кілька завдань. Так як вона генерується на етапі створення виконуваних файлів, в деяких випадках вона може бути використана для запобігання конфліктів просторів імен. Схожим чином в файли може вбудовуватися інформація про систему збирання, використаної при його створенні, організації-творця або автора. Крім того, в прикріплену підпис може бути додана інформація, що містить у собі додаткові мета-дані, такі як атрибути, торгові марки, версії використовуваних компонентів. [5]

Процес використовує криптографічні хеши, для валідування аутентичності, та цілісності. Підпис виконуваного коду забезпечує декілька важливих функцій. Найбільш використовунаю є безпека при процесі розповсюдження виконуваного коду. В деяких мовах програмування, він також може бути менізмом для забезпечення функціоналу простору імен. Майже кожен варіант підпису, включає в себе різні модифікації цифорових підписів для перевірки ідентичності авторів, та чек-сумму для перевірки того, що підписаний об’єкт не був змінений. Він також може бути використаний для передачі інформації про версійність продукту, або зберігати в собі необхідні мета данні для даного об’єкту.

Ефективність механізму підпису коду залежить від безпеки системи публічних відкритих систем. Безпека системи, як і інших систем типу PKI

залежить від того як виробники програмного забезпечення захищають свої приватні ключі він несанкціонованого доступу.

1.2 Безпека механізму підпису та сертифікації коду

Багато систем підпису виконуваного коду використовують метод підписання коду використовуючи пару ключів. Одни приватний, та один відкритий ключ. Наприклад при розробці використовуючи популярний фрейморк .NET, розробник автоматично підписує всі виконувані файли та бібліотеки кожен раз, коли компілює проект. Робить він це за допомогою ключа, який є унікальний для розробника, чи групи розробників або для кожного застосунку окремо. Розробник може згенерувати їх самостійно, або отримати його в центрі сертифікацій.

Найбільш корисним ця система є тоді коли джерело виконуваного файлу або скрипта може бути невідразу ідентифікаваним - наприклад Java аплети. Інший та не менш важливий випадок це безпека при розповсюдженні та оновленні програмного забезпечення для операційних систем. Windows, Mas OS, та більшість дистрибутивів Linux надають функціонал перевірки підпису коду щоб захиститися у випадках розповсюдження шкідливого програмного забезпечення під час оновлення, або патчинга операційних систем. Це дозволяє перевірити авторство коду, навіть якщо розповсюдження відбувається через фізичні носії (диски, флешки ітд).

Процес перевірки застосунків за допомогою центрів сертифікації

Під час створення виконуваного файлу, розробник самостійно може вирішити яким чином виконувати підпис застосунку. Найбільш розповсюдженим є отримання сертифікату в спеціальних центрах сертифікації. Публічний ключ який використовується для перевірки підпису повинен бути доступним в таких центрах, бажано щоб використовувалась система відкритих публічних ключів. Якщо користувач довіряє центру центру сертифікації, який

надає даний ключ, то він може довіряти легітимності коду, який був підписаний даним ключем. Майже всі операційні системи, та багато фреймворків мають вбудовану систему перевірки, а також мають декілька доступних центрів сертифікацій.

Тимчасові мітки

Тимчасові мітки в підписі проставляються довіреними організаціями в момент зміни даних у файлі. Особливо корисними ці позначки стають, коли сертифікат безпеки у деякого файлу було відкликано. Так, наприклад, якщо остання відмітка про зміну була проведена раніше, ніж сертифікат був відкликаний, валідація файлу може проходити успішно. Завдяки цьому близько 70% підписаних шкідливих файлів з відкликаним сертифікатом успішно використовуються, проходячи відповідні перевірки. Частина, що залишилася файлів виявляється підписаної з використанням вже недійсних сертифікатів. [7]

У всіх сучасних версіях Windows, починаючи з Windows XP SP2, при установці програмного забезпечення без такої цифрового підпису ви отримаєте попередження. Те ж саме до речі стосується і установки драйверів, які не мають відповідної цифрового підпису.

У разі, якщо цифровий підпис не знайдено, Windows видасть попередження, що у цієї програми «Невідомий видавець» і запускати її не рекомендується.

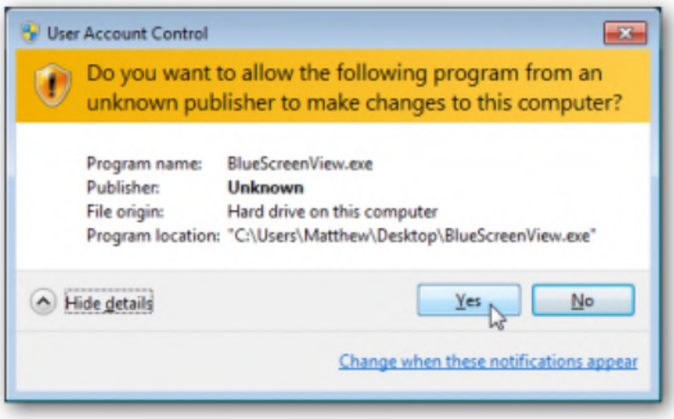


Рисунок 1.1 – Система перевірки сертифікатів Windows

У разі, якщо програма має цифровий підпис, то віконце буде виглядати інакше і ви також зможете подивитися інформацію про сертифікат.

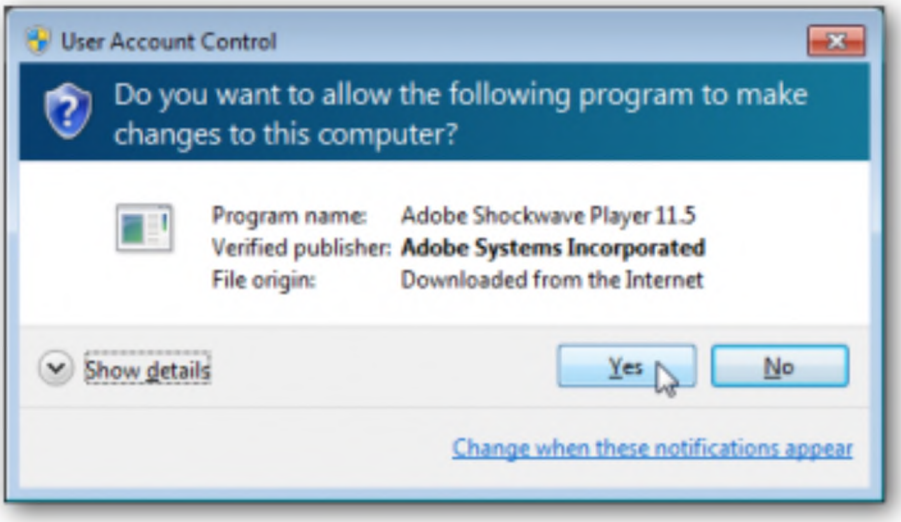


Рисунок 1.2 – Система перевірки сертифікатів Windows

1.3 Як працюють Code Signing сертифікати

Процес підпису коду схематично виглядає наступним чином:

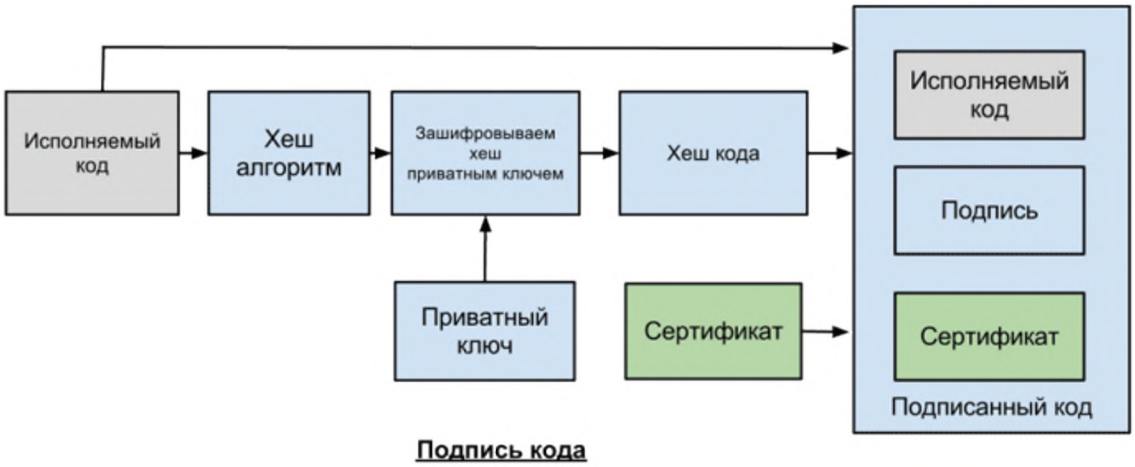


Рисунок 1.3 – Процес підпису коду схематично

1) Видавець (розробник) запитує Code Signing сертифікат у центру сертифікації

2) Використовуючи SIGNCODE.EXE або іншу утиліту для підпису коду видавець, Створити хеш коду, використовуючи алгоритми MD5 або SHA

3) Кодує хеш, за допомогою приватного ключа

4) Створює пакет, який включає в себе: код, зашифрований хеш і сертифікат видавця

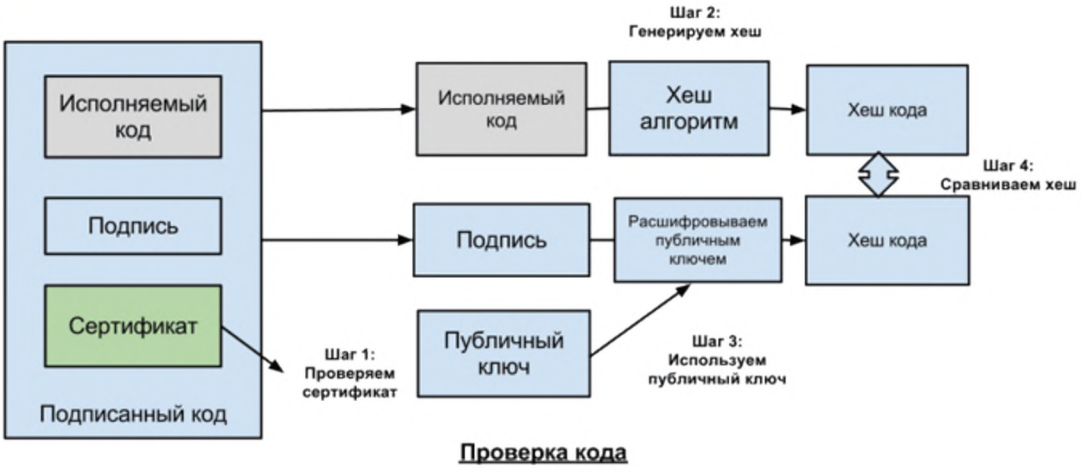


Рисунок 1.4 – Процес перевірки підписаного коду

1) Користувач завантажує або встановлює підписана ПО і платформа або система користувача перевіряє сертифікат видавця, який підписаний кореневим приватним ключем центру сертифікації
2) Система запускає код, використовуючи той же самий алгоритм створення хешу, як видавець і створює новий хеш
3) Використовуючи публічний ключ видавця, який міститься в сертифікаті, система розшифровує зашифрований хеш
4) І порівнює між собою 2 хешу

1.4 Детальніше про роботу центрів сертифікації

Коли розробник запитує цифровий сертифікат - центр сертифікації ідентифікує його і випускає сертифікат, пов'язаний з кореневим сертифікатом авторизації. Платформи і пристрої містять в собі кореневий сертифікат

відповідного центру сертифікації. Тобто якщо платформа або пристрій довіряє якомусь центру сертифікації, то воно довірять і вашому сертифікату, підписаного цим центром сертифікації.
У разі якщо хеши не збігаються ви отримаєте помилку при запуску такого ПО - це може означати, що ПО було модифіковано вірусом або зловмисником.

Коли ПО розшифровує цифровий підпис, воно перевіряє також кореневий сертифікат в системі, джерело перевіреної інформації. У разі використання самоподпісного сертифіката, ви отримаєте помилку: «видавець не може бути перевірений». Тому важливо використовувати сертифікати того центру сертифікації, чиї кореневі сертифікати вже встановлені в системі у передбачуваного користувача програми.

Timestamp або тимчасова мітка використовується для вказівки часу, коли цифровий підпис була зроблена. Якщо ця позначка присутня, то додаток, яке перевіряє підпис перевірить чи був сертифікат, пов'язаний з підписом дійсним на момент підпису. Якщо ж такої мітки немає, і термін сертифіката вже закінчився, то підпис буде вважатися недійсною.

Приклад:

Сертифікат дійсний з: 01.01.2014

Сертифікат дійсний до: 30.11.2016

Підпис зроблена: 05.08.2015

Підпис перевірена: 20.05.2018

С тимчасової міткою (timestamp) підпис пройде перевірку, оскільки на момент підпису сертифікат був дійсний. Без такої мітки сертифікат не пройде перевірку, оскільки на момент перевірки у сертифіката вже закінчився термін.

Тобто ця позначка дозволяє використовувати підписаний код, навіть після термін закінчення сертифіката.

1.5 Підпис виконуваного коду в контексті застосунків MacOS

MacOS Code Signing

Підписання коду - це технологія безпеки MacOS, яку ви використовуєте для підтвердження того, що програму було створено вами. Як тільки додаток буде підписано, система може виявити будь-які зміни в додатку - чи буде ця зміна введена випадково або через шкідливий код.

Ви берете участь у підписанні коду як розробнику, коли ви отримуєте ідентифікатор підпису та подаєте свій підпис на додатки, які ви доставляєте. Орган сертифікації (найчастіше Apple) підтверджує вашу особу підписування.

Примітка: У більшості випадків ви можете покладатися на підписування автоматичного коду Xcode, для цього потрібно лише вказати ідентифікатор підпису коду в налаштуваннях вашого проекту. Цей документ призначений для читачів, які повинні виходити за межі автоматичного підписання кодів - можливо, для усунення незвичної проблеми або для вживання інструмента codeign (1) у систему збірки. [8]

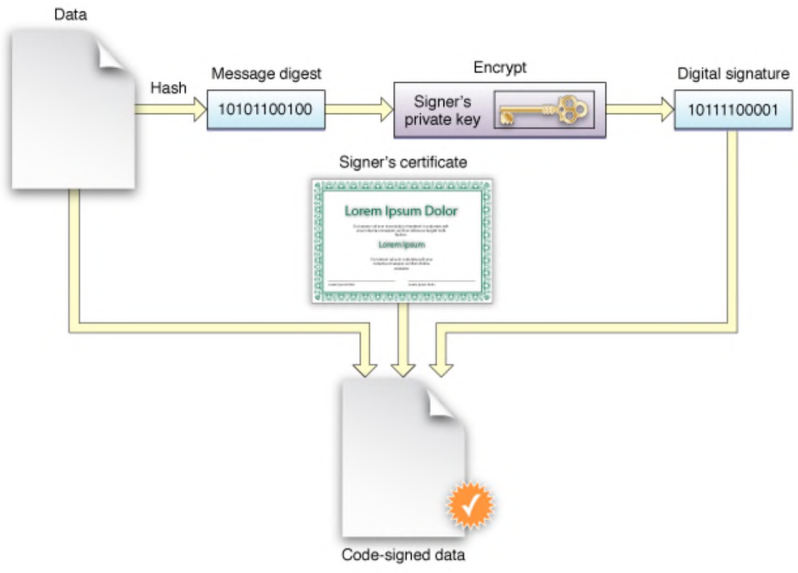


Рисунок 1.5 – Орган сертифікації Apple

Переваги використання технології підпису коду

Після встановлення нової версії програми, підписаною одним і тим же цифровим підписом, користувачеві не потрібно показувати сповіщення з проханням знову отримати дозвіл на доступ до keychain або подібних ресурсів. Поки нова версія використовує один і той же цифровий підпис, MacOS може ставитись до нової версії додатка так само, як і до останньої версії. Інші функції безпеки MacOS, такі як App Sandbox, також залежать від підпису коду. Зокрема, підписання коду дозволяє операційній системі:

- Переконатися, що фрагмент коду не було змінено з моменту його підписання. Система може виявити навіть найменші зміни, будь то навмисне (зміни зроблені зловмисником) або випадкове (як при пошкодженні файлу). Якщо цифровий підпис є незмінним, система може бути впевнена, що код також не був змінений.
- Визначення джерела коду (він містить ідентифікатор розробника). Підпис коду включає в себе криптографічну інформацію, яка однозначно вказує на конкретного автора.
- Визначати надані доступи для програми

Обмеження системи підпису коду

Цифровий підпис коду є лише одним з багатьох компонентів повного рішення безпеки, яке працює спільно з іншими технологіями та методами. Він не розглядає всі можливі проблеми безпеки.

Наприклад, підписування коду:

• не гарантує, що частина коду не містить уразливості системи безпеки.

• ну гарантує того, що додаток не завантажить додатковий небезпечний код або скрипт, наприклад, ненадійні плагіни під час виконання.

• не гарантує захист авторських прав, та захист від несанкціонованого копіювання, також ніяким чином не змінює та не приховує вміст виконуваного файлу.

Структура сертифікату підпису виконуваного коду

Можна підписати будь-який коди, включаючи інструменти, програми, скрипти, бібліотеки, плагіни та інші "кодові" дані. Крім того, ви можете підписати програми інсталлери. У всіх випадках цифровий підпис коду складається з трьох частин:

• Пломба. Це сукупність контрольних сум або хешей різних частин коду, створених програмним забезпеченням підпису коду. Пломбу можна використовувати під час перевірки для виявлення змін.
• Цифровий підпис. Програма підпису коду зашифровує печатку за допомогою ідентифікатора підписувача для створення цифрового підпису. Це гарантує цілісність печатки.
• Коди вимоги. Це правила, що регулюють перевірку підпису коду.

Пломба

Механізм цифрового підпису генерує пломбу обраховуючи хеш кожного вашого файлу, включаючи виконуваний файл, plist.file, ресурси та бібліотеки.

Цифровий підпис

Підписаний код може містити кілька різних цифрових підписів:
• Якщо бінарний файл універсальний (fat-binaries) , то кожна частина такого виконуваного файлу підписується окремо. Цей підпис зберігається в самий двійковий файл.

- Також підписуються всі ресурси які знаходяться всередині бандлу (наприклад, файл Info.plist, якщо є). Ці підписи зберігаються у файлі під назвою _CodeSignature / CodeResources в директорії.
- також підписуються всі бібліотеки та допоміжні бінарні файли, які теж можуть знаходитись в средині бандлу [10]

1.6 Перевірка підпису коду різними підсистемами MacOS

Система перевіряє виконуваний код декілька разів та по різним причинам. Для цього існує декілька підсистем, які виконують перевірку на різних етапах виконання файлу. До таких підсистем можна віднести: App Sandbox, Gatekeeper, Application Firewall, Parental Controls, Keychain Access Controls, Developer Tools Access

Таблиця 1.1 наводить конкретні приклади того, як підписи коду використовуються різними підсистемами в MacOS, щоб забезпечити застосування політик довіри, специфічні для певного виду системного ресурсу. Зауважте, що це типова поведінка; ви можете змінити багато політик підписування коду macOS за допомогою команди spctl (8).

Наведені вище приклади демонструють, як рішення щодо доступу визначаються конкретними підсистемами, а не самим підписом коду. Крім того, вони висвітлюють різноманітність політик.

Таблиця 1.1 - Порівняння характеристик підсистем MacOS

Підсистема	Основна функція	Політика надання
Application Sandbox	Обмежує доступ до системних ресурсів в залежності від прав застосунку	Дозволяє доступ до визначеного ресурсу, якщо дозвіл для нього є сертифікаті підпису

Підсистема	Основна функція	Політика надання
Gatekeeper	Забороняє користувачу запускати виконуваний файл від не перевіриного розробника	Перевірка полів Developer ID або Mac App Store
Application Firewall	Обмежує вхідний та вихідний доступ для використання мережі	Дозволяє якщо є наявна відмітки про доступ. В іншому випадку запитує користувача.
Parental Controls	Обмежує використання програм для відповідних користувачів системи	Працює тільки за налаштуваннями адміністратора
Keychain Access Control	Контролює доступ застосунку до чітко визначених елементів, які можуть зберігатися в системі Keychain	Створююча програма автоматично отримує доступ до своїх елементів та визначає політику доступу для інших застосунків (використовуючи систему перевірки сертифікатів)
Developer Tools Access	Визначає які саме програми, можуть викликати інструменти які використовуються для розробки та аналізу застосунків	Дозволяє якщо є наявна відмітки про доступ.

Наприклад:

- У DTA немає політики відстеження. Він запитує доступ кожен раз, коли виконується файл. Йому немає необхідності зберігати запит на диску.
- Application Firewall використовує сертифікат підпису як для встановлення початкових доступів, так і для того щоб відслідковувати їх надалі
- Application Sandbox надає права на ресурси в залежності від вбудованих в сертифікат запитів. Наприклад доступ до мережі, камери, частини файлової системи тощо. Фактично всі права (для додатків які знаходяться в Application Sandbox) зберігаються та закріплюються в сертифікаті підпису, тому будь-яке додаток, який використовує будь-які ресурси системи, вимагає підпису коду.

Деякі частини macOS не вимагають ідентичність підпису. Вони вимагають лише саму наявність сертифікату підпису. Прикладами такого використання є система Keychain та Parental Control. Самостійно підписані

особисті дані та власні сертифікати (CA) за замовчуванням працюють у цьому випадку. Оскільки вони не працюють для системи Gatekeeper, вони, як правило, не рекомендуються для розповсюдження, але можуть бути корисними під час розробки або для тестування.

Інші компоненти macOS дозволяють використовувати лише ті сертифікати, які взяті з органів сертифікації. Для цих систем важливо враховувати автора виконуваного коду, те що він був зареєстрований. Приклад Application Firewall - один з прикладів. Самостійно підписані сертифікати недійсні для цієї системи перевірки, якщо тільки вони не були додані в систему користувачем власноруч. Це можуть зробити тільки ті користувачі, які мають права адміністратора. [9]

Здебільшого рішення щодо доступу до елементів приймається один раз, що може вплинути на поведінку системи. Наприклад, шкідливий код, який можна виконати через проблему переповнення буфера, може виконуватись, оскільки він не є чатиною програми в час старту виконуваного файлу і, таким чином, не перевіряється системою Gatekeeper. [11]

Карантин файлів як частина механізму перевірки сертифікатів

Починаючи з версії Mac OS X 10.5 Leopard, Apple додав функцію під назвою карантин файлів. Вона визначила спеціальні метадані, які були додані до файлу під час завантаження його з Інтернету. Коли користувач намагався відкрити виконуваний файл (наприклад програму або скрипт), який був позначений як файл з карантину, система запитує у користувача чи дійсно потрібно відкривати даний файл. Ця перевірка є справедливою на першому рівні захисту користувача від файлів які є скачані з невідомих ресурсів та можуть являтися не тим, що очікує користувач. Проблемою такої перевірки є те,

що користувачі часто ігнорують будь якого типу повідомлення, і намагаються найшвидше їх закрити.

Іншим обмеженням карантину файлів було те, що він працював лише з файлами, завантаженими з Інтернету, та ігнорував файли які копіювали з компакт-диска чи флеш-накопичувача, зовнішнього жорсткого диска. Крім того, карантин працював лише тоді, коли файл був завантажений за допомогою програми, що підтримує функціонал карантину файлів, наприклад Safari.app або Mail.app. Більшість сторонніх програм для завантаження файлів через Інтернет також підтримують карантин, але деякі - ні.

Окрім цього, Apple продовжувала працювати не тільки дан карантином, наприклад, вони додали базовий захист від шкідливих програм у формі технології XProtect. Тут розглядаються файли, які були поміщені в карантин після їх відкриття, і представляє користувачеві попередження, якщо файл, насправді є відомим шкідливим програмним забезпеченням. Звичайно, це також страждає слабкими сторонами. Такий тип антивірусного програмного забезпечення захищає лише від відомих вірусів, і ніяким чином не захищає від нових невідомих вірусів.

Система перевірки та запуску застосунків Gatekeeper

Починаючи з версії Mac OS X 10.8 Mountain Lion, Apple додав ще одну нову функцію, побудовану над системою карантину файлів. Ця функція, яку вони назвали “Gatekeeper", функціонує по-іншому коли користувач намагається запустити застосунок з карантину.

За замовчуванням він може користуванися додатками які були скачані тільки з магазину AppStore Тепер. Змінити налаштування користувач може в Preferences -> Security -> General, також для цього потрібні права адміністратора. Вданий час є три опції, показані справа.

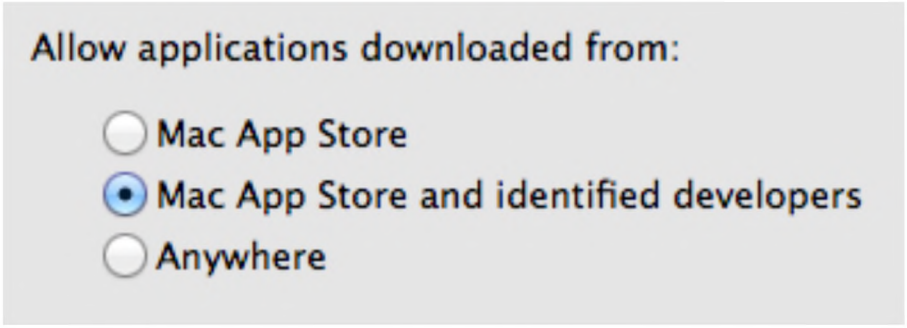


Рисунок 1.6 – Налаштування Gatekeeper

Кожна опція Gatekeeper забезпечує різний рівень безпеки.

Перша дозволяє користувачеві використовувати додатки, завантажні лише з магазину App Store. Оскільки Apple аналізує кожну програму в магазині App Store і розповсюджує лише додатки, що відповідають всім правилам, це найбезпечніший варіант. Але це найбільш обмежений варіант, тому що не всі застосунки які можуть бути корисними для користувача, можуть пройти правила перевірки.

Друга опція дозволяє запускати додатки, які "підписані" зареєстрованим розробником. Це означає, що розробник зареєстрував в системі Apple Developer та сплатив плату. Натомість розробник отримує захищений сертифікат, який може бути використаний для підписання коду додатка. Хоча це і не є неможливим, але є дуже малоймовірно, щоб розробник використовуючи свій обліковий запис, який був куплений та оплачений, а відповідно пов'язаний з контактною інформацією та платіжною інформацією, використає його для створення шкідливого програмного забезпечення. Тим більше, що Apple може просто та швидко відкликати сертифікат і Gatekeeper заблокує додаток на всіх комп'ютерах.

Останній варіант дозволяє користувачеві відкривати додатки, завантажені з будь якого джерела. Це найменш безбечний варіант, він повністю вимикає роботу Gatekeeper. На щастя, ніколи не потрібно обирати цей параметр,

оскільки ви можете відмовитися від цього захисту в кожному окремому випадку.

Підсистема Path Randomization, та випадки її використання

Розробники можуть підписати образ диску (таким чином розповсюджуються застосунки за межами магазину Mac App Store), які система також може перевіряти на наявність підпису розробника. Це дозволяє розробникам гарантувати цілісність контенту який знаходиться всередині образу. Крім того, "рандомізація шляху" виконує застосунки з випадкового прихованого шляху та не дозволяє їм отримувати доступ до зовнішніх файлів відносно їх розташування.

Для прикладу, якщо застосунок знаходиться за шляхом: “~/Downloads/Example.app” то запускатися він буде з випадково згенерованого шляху формату “/private/var/temp/*……*/Example.app” Ця функція вимикається, якщо застосунок з підписаного образу диска встановлено або користувач самостійно перемістив образ диска до каталогу Applications.

Отже, тепер, коли ми маємо базові уявлення про те, як Gatekeeper працює для захисту від шкідливих програм, ми повинні зрозуміти його недоліки. Gatekeeper побудований на основі карантину файлів, тому проблеми карантину файлів є проблемами системи Gatekeeper. Gatekeeper може блокувати програми, які були поміщені в карантин, тоді як будь-які додатки, які не були позначені карантином, зможуть відкриватися необмежено. Він також не буде функціонувати, якщо повністю відключити карантин файлів.

Важливо розуміти що Gatekeeper не може нічого зробити якщо говорити про шкідливе забезпечення яке використовує наприклад проблеми Java. Оскільки це відбувається через помилку в Java, вона обходить всю систему карантину а відповідно і Gatekeeper.

Gatekeeper не є досконалим, і ніколи не був. Важливо розуміти, що, як і будь-якої системи безпеки, ефективність Gatekeeper обмежена.

Це не означає, що він слабкий або не корисний, однак навпаки, Gatekeeper - надає базовий рівень захисту. Були інші види зловмисного програмного забезпечення, які були повністю заблоковані Gatekeeper, якщо користувач не вимкнув його.

Висновки до розділу 1

В даному розділі було проаналізовано принципи роботи систем перевірки сертифікатів. Описано всі етапи роботи механізму перевірки сертифікатів, їх роботу з центрами сертифікацій. Досліджено будосу Codesigning сертифікатів операційної системи MacOS. Детально розглянуто існуючі системи перевірки сертифікатів. Досліджено підсистеми мезанізму, а саме такі як Gatekeeper, Path-randomization.

2 ПРИНЦИП РОБОТИ СИСТЕМ ПЕРЕВІРКИ СЕРТИФІКАЦІЇ НА MACOS

2.1 Загальна характеристика механізму

В ОС macOS використовується технологія Gatekeeper, яка забезпечує запуск тільки довіреної програмного забезпечення на комп'ютері Mac.

Рисунок 2.1 – Структурна схема використаття механізмав сертифікації MacOS

Найбезпечнішим місцем для завантаження програм на жорсткий диск комп'ютера Mac є магазин App Store. Компанія Apple перевіряє кожну програму в App Store, перш ніж прийняти, і підписує її, щоб вона не піддалася несанкціонованого втручання або зміни. Якщо з програмою виникне проблема, компанія Apple може швидко видалити її з магазину.

Якщо ви завантажуєте програми з Інтернету або безпосередньо на веб-сайті розробника і встановлюєте їх, ОС MacOS продовжує захищати комп'ютер Mac. Коли ви встановлюєте на комп'ютер Mac програми, плагіни і настановні пакети не з App Store, ОС MacOS перевіряє підпис ідентифікатора розробника і стан підтвердження справжності, щоб переконатися, що програмне забезпечення випустив ідентифікований розробник, і що воно не змінено. В ОС macOS Mojave розробники можуть також підтвердити справжність своєї програми у Apple. Для цього перед розповсюдженням програму потрібно відправити в Apple, щоб вона пройшла перевірки безпеки.

Принципи роботи системи Gatekeeper

Безпечне відкриття програм на комп'ютері Mac

В ОС macOS використовується технологія Gatekeeper, яка забезпечує запуск лише надійного програмного забезпечення на комп'ютері Mac.

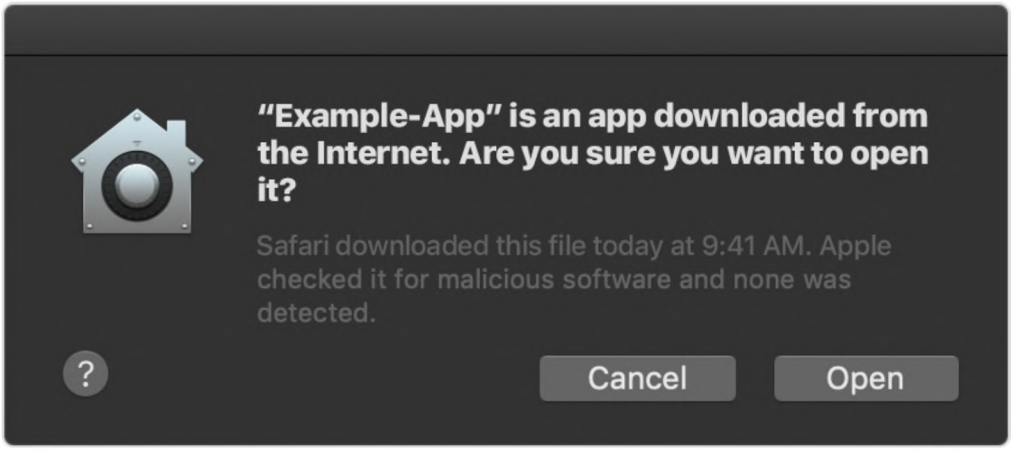


Рисунок 2.2 – Безпечне відкриття програм на комп'ютері Mac

Самим безпечним місцем для завантаження програм на комп'ютер Mac є магазин App Store. Компанія Apple перевіряє кожну програму в App Store, перш ніж прийняти, і підписує її, щоб вона не піддалася несанкціонованому втручанню або зміні. Якщо з програмою виникають проблеми, компанія Apple може швидко видалити її з магазину, тим самим зупинити її розповсюдження.

Якщо ви завантажуєте програми з Інтернету або безпосередньо на веб-сайті розробника і встановите їх, OS MacOS продовжує захищати комп'ютер Mac. Коли ви встановите на комп'ютер Mac програми, плагіни та установчі пакети не з App Store, OS MacOS перевіряє підпис ідентифікатора розробника та підтвердження автентичності, щоб переконатися, що програмне забезпечення випустило ідентифікований розробник, і що він не змінений. В ОС MacOS Mojave розробники також можуть підтвердити автентичність своєї програми в Apple. Для цього перед розповсюдженням програми необхідно відправити в Apple, щоб вона пройшла перевірку безпеки.

2.2 Налаштування безпеки застосунків MacOS

За замовчуванням налаштування безпеки та конфіденційності на комп'ютері Mac дозволяють встановлювати програми лише з магазину App Store і від ідентифікованих розробників. Для додаткової безпеки ви можете дозволити встановлення програм тільки з магазину App Store.

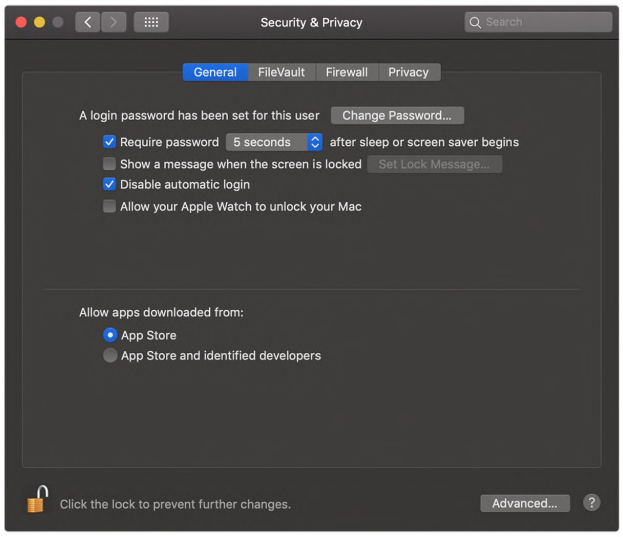


Рисунок 2.3 –Налаштування безпеки операційної системи MacOS

В меню «Системные настройки» натисніть «Защита и безопасность» і виберіть «Основні». Натисніть значок замка та введіть пароль, щоб внести зміни. Виберіть App Store в розділі «Дозволити використання програм, завантажених з».

Відкриття програми з підписом розробника або підтвердженням справжності

Якщо на вашому комп'ютері Mac налаштований дозвіл на установку програм з App Store і від ідентифікованих розробників, при першому запуску програми від ідентифікованого розробника комп'ютер Mac видасть запит про те, чи дійсно ви хочете її відкрити.

Для програми з підтвердженням справжності від Apple буде вказано, що вона пройшла перевірку справжності:

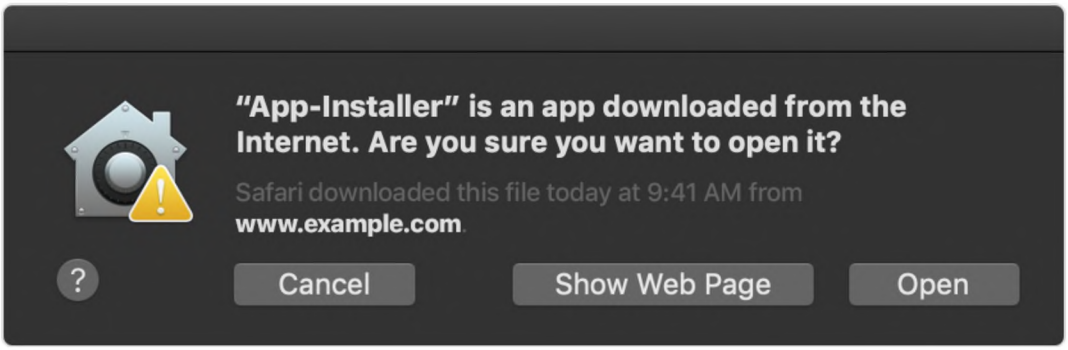


Рисунок 2.4 – Вікно системи Gatekeeper

Якщо ОС macOS виявляє шкідливу програму

Якщо ОС macOS виявляє проблему в програмі - наприклад, наявність шкідливого вмісту або модифікацій, внесених після перевірки, - ви отримаєте повідомлення при спроби відкрити програму і запит на перенесення її в кошик.

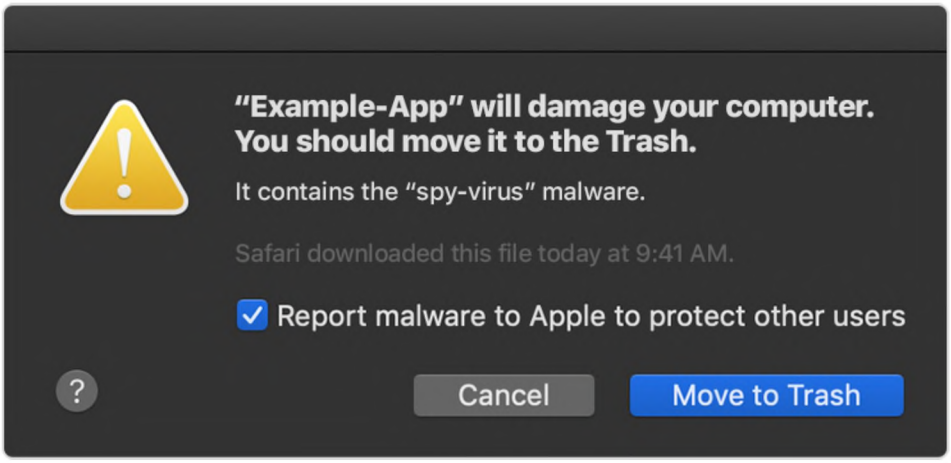


Рисунок 2.5 – Вікно системи Gatekeeper

Якщо відображається повідомлення з попередженням і не вдається встановити програму

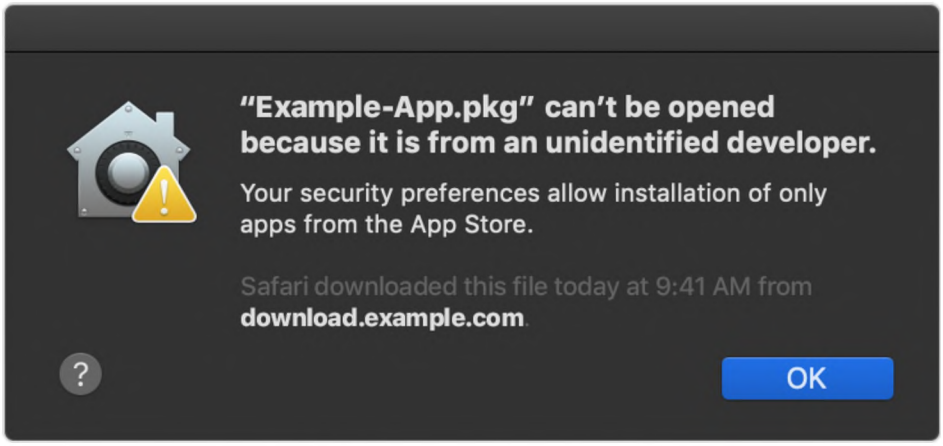


Рисунок 2.6 – Вікно системи Gatekeeper

Якщо ви налаштували на комп'ютері Mac дозволи встановити лише програми з App Store і намагаєтеся встановити програму з іншого місця, комп'ютер Mac видає повідомлення, що ця програма не з App Store.

Якщо на компьютере Mac дозволено встановлення програм з магазину App Store і від ідентифікованих розробників, і ви намагаєтеся встановити програму, яка не зареєстрована в компанії Apple ідентифікованим розробником, також відобразиться попередження.

Ці повідомлення не обов'язково означають, що з програмою чогось не так. Наприклад, деякі програми були створені до реєстрації ідентифікаторів розробників. Якщо ви видно попередження, це означає, що програма не була підписана розробником, тому OS macOS не може перевірити, чи була вона модифікована або розірвана після випуску.

Можливо, краще знайти більш нову версію програми в App Store або іншу програму.

2.3 Встановлення програм від неідентифікованого розробника

Якщо ви впевнені, що програма, яку ви хочете встановити, з достовірної джерела і не містить несанкціонованих змін, ви можете тимчасово перезаписати налаштування безпеки комп'ютера Mac і відкрити її.

В Finder щелкните програму, утримуючи натиснутою клавішу Control, оберіть в меню «Відкрити», потім у діалоговому вікні натисніть кнопку «Відкрити».

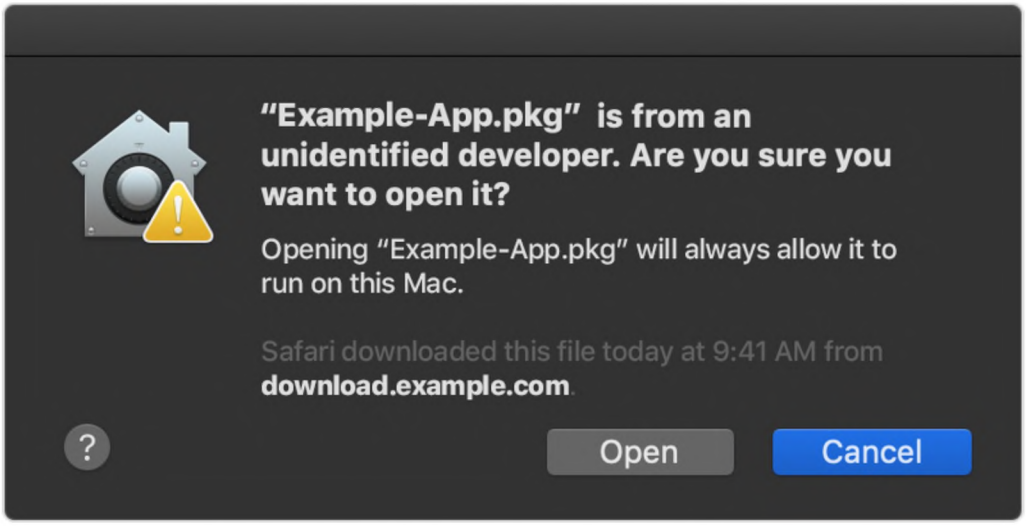


Рисунок 2.7 – Вікно системи Gatekeeper

Введіть ім'я та пароль адміністратора при відображенні відповідного запиту.

Програма буде збережена як виключення з налаштувань безпеки, і ви зможете відкрити її двома кліками, як і будь-яка інша авторизована програма.

2.4 Структура виконуваних файлів в операційній системі MacOS та інструменти їх дослідження

З метою поглибленого вивчення операційної системи Mac OS та вивчення методів захисту ПЗ, потрібно в першу чергу розібратися із структурою тих файлів які ми будемо намагатися захистити. Це питання актуальне тому, що більшіть розробників не знають як влаштовані такі файли. Знати їх структуру, те як вони працюють необхідно для того, щоб розуміти яким саме чином зловмисники змінюють ці файли для своїх потреб.

Далі розглянуто структуру виконуваних файлів, порядок їх загрузки, зв'язування з іншими бібліотеками, порядок їх виконування. А також розглянуто програмні інструменти за допомогою яких можна досліджувати, змінювати, дизасемблювати такі файли.

Утиліти для аналізу програм

Для аналізу програм існує два основних підходи: динамічний та статичний. Динамічний аналіз передбачає запуск програмного коду під дебагером або у віртуальній середовищі та аналіз його поведінки. Статичний аналіз програм - це дослідження програмного коду за допомогою дизассемблера без запуску коду.

Який підхід краще застосовувати залежить від конкретної ситуації. Обидва методи - не взаємовиключають один одного, а часто доповнюють один одного.

Утиліти для динамічного аналізу програм

Як і більшість Unix-систем, Mac OS надає багато корисних утиліт, які допомагають при динамічному аналізу прикладних програм та діагностики системи. Багато з них зпозичені на Mac из Unix, але є й такі, які були розроблені спеціаольно для Mac OS.

Всі утиліти можна розбити на дві категорії.

1) Утиліти, які використовуються для дослідження процесів:

• fs_usage - надає інформацію про системні виклики відносно активності по відношенню до файлової системи;

• heap - перечислює всі блоки пам'яті, виділені в динамічній пам'яті окремим процесом;

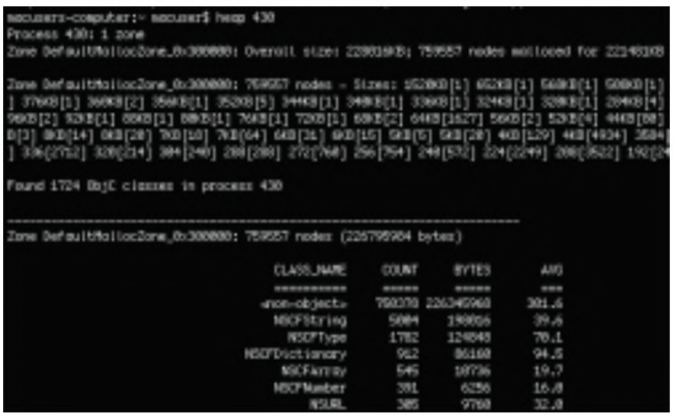


Рисунок 2.8 – Використання вбудованої утиліти heap

- *lsof* — відображає файли, відкриті в різних процесах;

- *top* — відображає файли, відкриті в різних процесах;

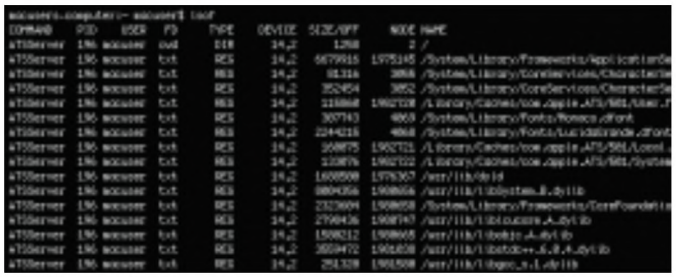


Рисунок 2.9 – Використання вбудованої утиліти *top*

• vm_stat - відображає статистику використання системи віртуальної пам'яті;

• gdb - дебагер, дозволяє відладжувати програми;

• ddb - дебагер ядра, вимагає підключення через послідовний порт;

• ktrace - служить для відстеження інформації про системні події на рівні ядра для зазначеного процесу;

• kdump - відображає інформацію, створену програмою ktrace;

• sc_usage - відображає статистику вказаного процесу, такі як використання процесорного часу, використання системних викликів і т. Д.

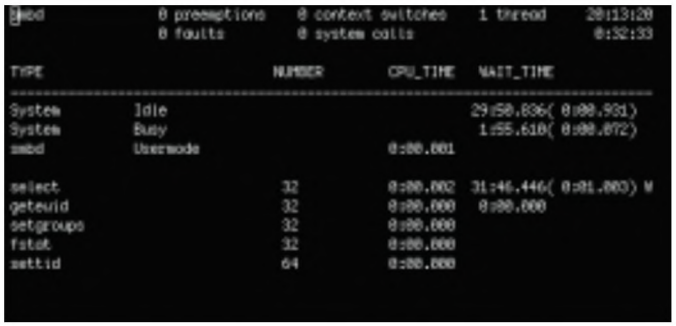


Рисунок 2.10 – Використання вбудованої утиліти sc_usage

Мережеві інструменти

Перераховані нижче мережні утиліти добре відомі в світі Unix:

Рисунок 2.11 – Використання вбудованої утиліти netstat

• netstat - надає різні дані, що відносяться до мережевої підсистеми;

• tcpdump - відображає мережевий трафік.

Для Mac OS X доступні і багато інших знайомім користувачам Unix мережевих інструментів, таких як nmap та WireShark.

Необхідно відзначити, що більшість програм з відкритими вихідними кодами, які існують під Unix, можуть бути легко зібрані під Mac OS X. Досвідчений Unix-користувач може створити собі робоче середовище, яке буде мало що відрізняється від звичайного для нього Unix.

Інструменти для статичного аналізу

Дуже часто при аналізі шкідливих програм відсутня можливість запускати досліджувану програму, або запуск програмного коду небажано через загрози безпеки. Тому для аналізу досліджуваної програми необхідно використовувати дизасемблювання файлів.

Основною утилітою для аналізу файлів Mach-O формату є програма otool. З її допомогою можна отримати таку інформацію, як заголовок файлу, завантажувальні команди, вхід точки і навіть дизассемблировать вміст секції, що містить виконуваний код.

- *otool* — використовується для аналізу mach-o файлів;

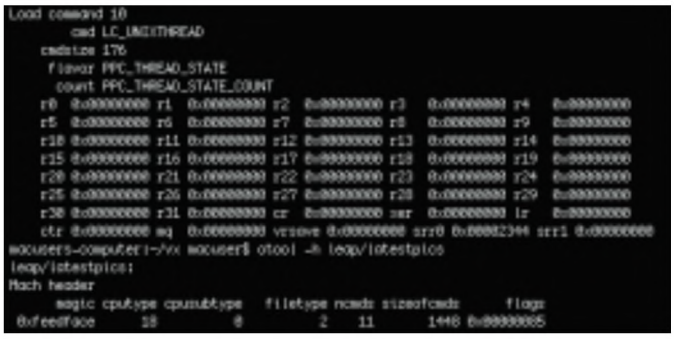


Рисунок 2.12 – Використання вбудованої утиліти otool

- *file* — визначає тип файлу;

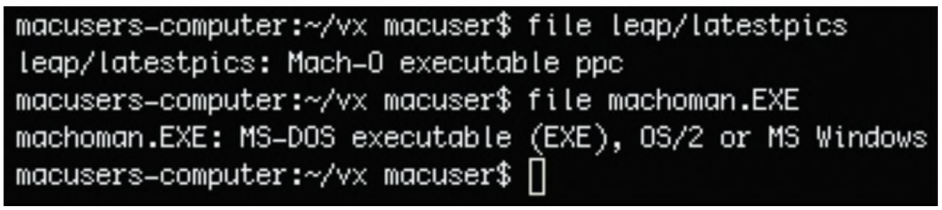


Рисунок 2.13 – Використання вбудованої утиліти file

- *xxd* — дозволяє здійснити перетворення бінарного файлу в шестнадцяткове представлення і назад;
- *IDAPro* — дизасемблер.

Використані інструменти дослідження

codesign -- Створення та керування підписами

Команда **codesign** використовується для створення, перевірки та відображення підписів коду, а також для запиту динамічного статусу підписаного коду в системі.

codesign вимагає один аргумент операції яка буде виконуватись, і будь яку кількість модифікаторів команди. Може виконуватись над будь-якою кількістю файлів але для всіх них виконує одну й ту саму операцію.

Приклади використання команди codesign:

Щоб підписати застосунок Example.app потрібно виконати команду:

codesign -s authority Example.app

Щоб перевірити підписExample.app і надати деякий докладний вихід:

codesign --verify --verbose Example.app

Щоб перевірити запущений процес pid = 666:

codesign --verify 666

Щоб відобразити всю інформацію про підпис коду Example.app:

codesign --display --verbose=4 Example.app

Витягнути внутрішні вимоги від Example.app до стандартного виводу:

codesign --display -r- Example.app

Otool - інструмент для відображення вмісту файлів. Консольна команда otool відображає вибрані частини виконуваних файлів, бібліотек. Використовується вона за наступним синтаксисом

otool [option ...] [file ...]

Так як ми досліджуємо Mach-O файли, то нам потрібно звернути увагу на наступні опції цього інструменту:

-h Відобразити Mach заголовок. (рисунок 1.1)

```
MacBook-Pro-Roma:~ roma$ otool -h /Users/roma/Desktop/ButtonBinding2
/Users/roma/Desktop/ButtonBinding2:
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
 0xfeedfacf 16777223          3  0x80           2    21       2568 0x00200085
```

Рисунок 2.14 – Результати команди відображення header файлу

-l Відобразити команди загрузки.

-L Відобразити імена та версії бібліотек які використовуються в даному файлі (рисунок 2.15).

```
MacBook-Pro-Roma:~ roma$ otool -L /Users/roma/Desktop/ButtonBinding2
/Users/roma/Desktop/ButtonBinding2:
        /System/Library/Frameworks/Foundation.framework/Versions/C/Foundation (compatibility version 300.0.0, current version 1258.0.0)
        /usr/lib/libobjc.A.dylib (compatibility version 1.0.0, current version 228.0.0)
        /usr/lib/libSystem.B.dylib (compatibility version 1.0.0, current version 1226.10.1)
        /System/Library/Frameworks/AppKit.framework/Versions/C/AppKit (compatibility version 45.0.0, current version 1404.46.0)
        /System/Library/Frameworks/CoreFoundation.framework/Versions/A/CoreFoundation (compatibility version 150.0.0, current version 1258.1.0)
```

Рисунок 2.15 – Відображення команд загрузки виконуваного файлу

-s segname sect name Відобразити контент вибраного сегменту та секції

(__OBJC,__protocol), (__OBJC,__string_object)(__OBJC,__runtime_setup)

-t Відобразити контент секції (__TEXT,__text) При наявності флагу -v також дизасемблюється текст.

-d Відобразити контент секції (__DATA,__data).

-o Відобразити контент секції __OBJC який використовується Objective-C під час виконання програми.

Для більш детального використання і опису команд перегляньте інформацію про каманду в “man otool”. До переваг цього додатку можна віднести те що він нативний, а одже в разі змін підтримується розробником.

MachOView - зручний додаток, який розповсюджується по GPL (універсальна вільна ліцензія). Додаток має графічний інтерфейс та зручний у використанні. (рисунок 2.16)

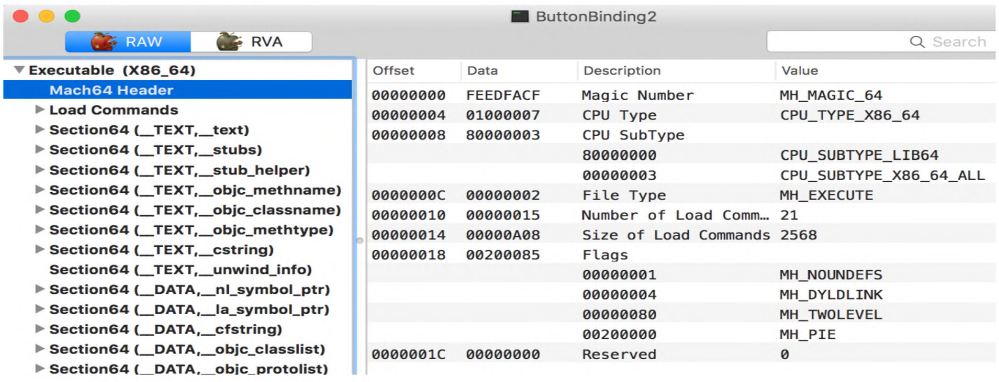


Рисунок 2.16 – Графічний інтерфейс MachOView

Hopper Disassembler - це програма дизасемблер для платформи OS X та Linux, за допомогою якої можна декомпілювати застосунки написані під системи 32/64bits Intel Mac, Linux, Windows and iOS (рисунок 1.4). Зворотна розробка (англ. reverse engineering) — дослідження деякого пристрою чи програми з метою розуміння принципів роботи досліджуваного об'єкта.

Найчастіше використовується з метою створення об'єкта, за функціональністю аналогічного досліджуваному але без точного копіювання його функцій [9]. За допомогою Hopper можна виконувати всі базові методи реверс-інжинірингу, а саме:

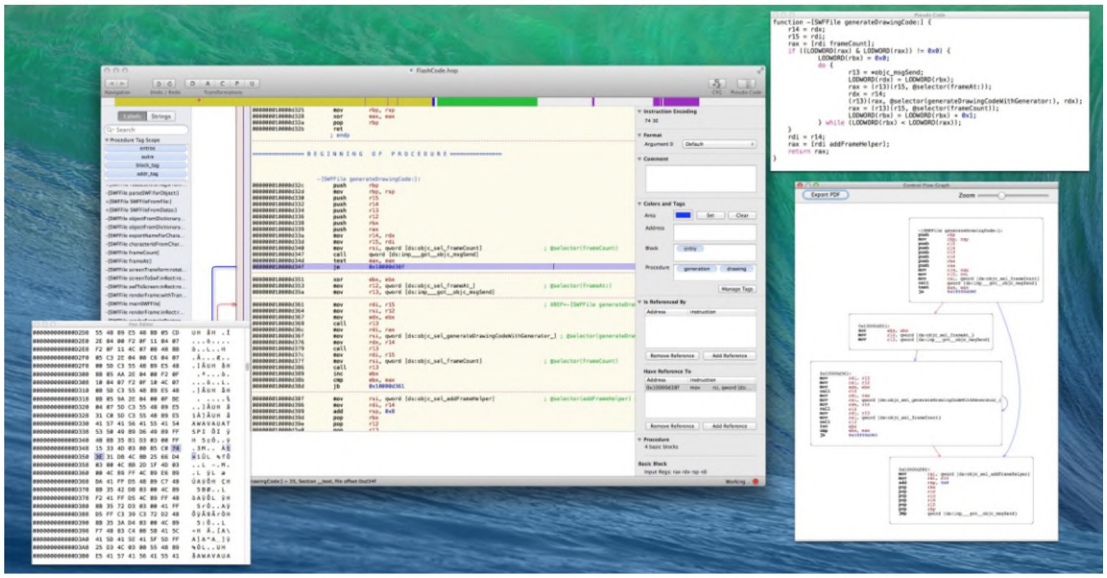


Рисунок 2.17 – Графічний інтерфейс дизасемблера Hopper

Моніторинг активності. Таким способом найчастіше проводиться дослідження протоколів обміну інформацією. Наприклад для дослідження мережевих проколів може використовуватися перехоплення потоків даних в мережі за допомогою спеціалізованих програмних чи/та апаратних засобів. Цей метод може не дати повного уявлення про алгоритми функціонування ПЗ.

Дизасемблювання. Машинний код програми читається та перекладається мовою асемблера для свого розуміння в чистому вигляді. Таким способом можна досліджувати будь-яке програмне забезпечення, але за допомогою використання певних технологій при розробці ПЗ дизасемблювання можна значно ускладнити. Метод вимагає високої кваліфікації людини, що проводить зворотну розробку та великих затрат часу.

Декомпіляція. Полягає у перекладі машинного коду програми мовою високого рівня. Метод важко реалізувати з огляду на складність створення інструментів.

Дослідження структури виконуваних файлів mach-o

Не беручи до уваги специфічні виконувальні формати, основним форматом виконуваних файлів в операційній системі OS X є файли формату Mach-O. Як каже офіційна документація Mach-O скорочення від Mach object file format -це формат виконуваних файлів, коду, статичних та динамічних бібліотек. Він забезпечує більш зручний та швидший метод доступа до інформації символьної таблиці. Mach-O використовується майже всіма системами заснованих на Mach ядрі такі як NeXTSTEP, OS X, iOS, систем які використовують цей формат як нативний.

Mach-O - це не якийсь специфічна структура данних, а лише впорядкований бінарний набір зрупований на частини данних.

Арітекура OS X визначає всі потрібні їй компоненти під час виконування, так звана runtime архітектура, саме під ці вимоги було створено Mach-O файли.

Відповідно до офіційної документації Mach-O файл складається з трьох частин (рисунок 2.18):

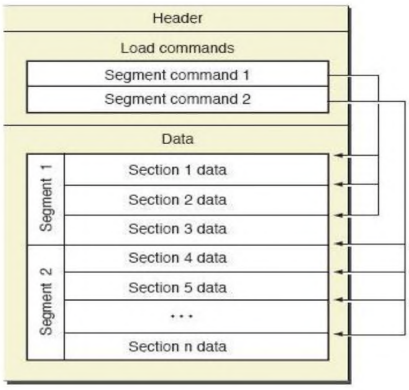
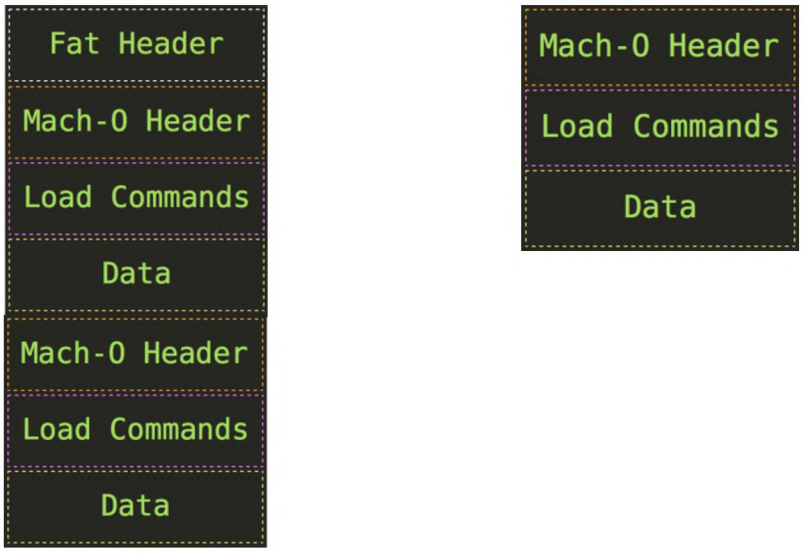


Рисунок 2.18 – Схематичне зображення структури виконауваного файлу Mach-O

1) Header – заголовок, який містить основну інформацію про цільову архітектуру порядок виконування, мета-інформацію таку я кількість виконуваних команд, розмір частини, опції інтерпретації файлу.

2) Load Commands - команди загрузки, які повідомляють як і куди загружати частини сегментів, також містить таблицю символів а також список всіх бібліотек від яких залежить виконуваний файл, для того щоб перед виконуванням спотку загрузити їх в пам'ять.

3) Data - сегменти, зазвичай найбальша частина файлу яка містить весь виконуваний код. Описують регіони пам'яті, куди загружати секції з кодом або данними.



а) б)

Рисунок 2.19 – Зображення структури виконуваних файлів Mach-O а) Universal Binaries б) Mach-O

Взагалі є два типа виконуваних файлів для OS X: Mach-O (рисунок 1.6 а) та Universal Binaries (рисунок 2.19 б) часто їх ще називають Fat files. Різниця між ними тільки в тому що Mach-O містить в собі код який може виконуватись тільки на одній з архітектур (i386, x86_64, arm64 та інші), тим часом коли Fat files містять в собі декілька файлів для різних архітектур [4]. Їх структура досить проста і відрізняється наявністю Fat Header - заголовок який визначає декілька Mach-O файлів в собі.

Структура заголовку досить проста, наведемо приклад та розміщення цієї структури в пам’яті виглядає наступним чином (рисунок 2.20):

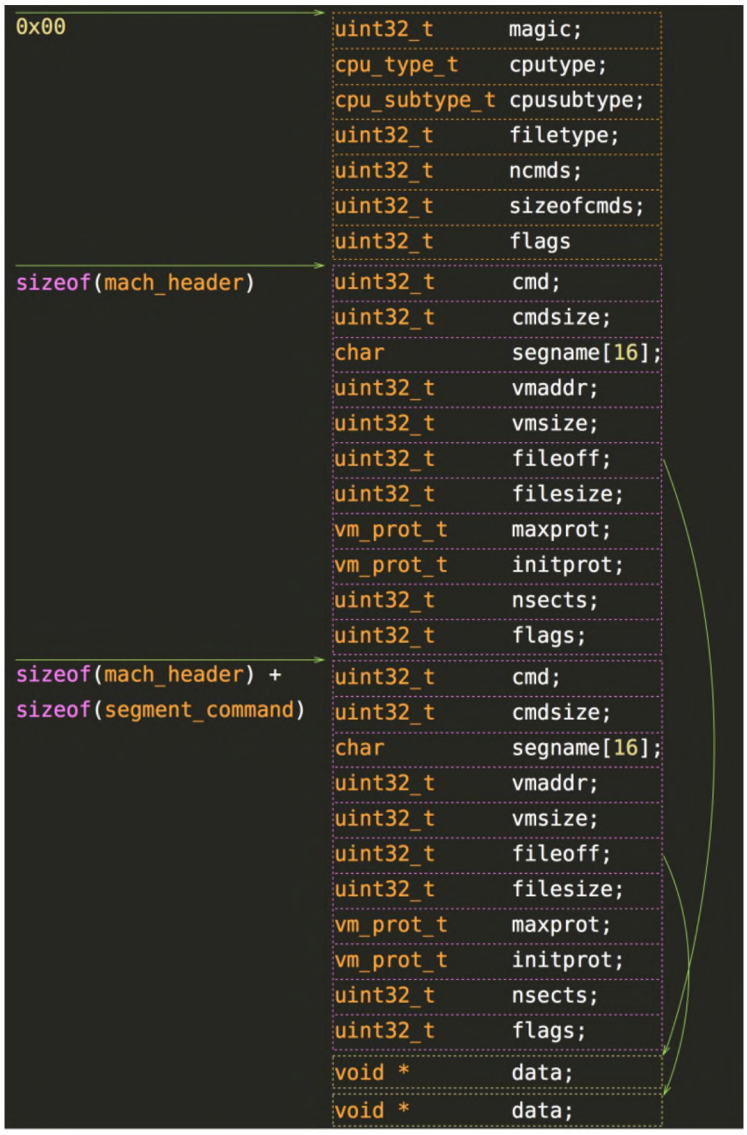


Рисунок 2.20 – Структура заголовку виконуваного файлу Mach-O

Це означає, що для того щоб прочитати потрібні данні з файлу потрібно лише знати коректне структуру файлу, тип змінної яку ми хочемо прочитати та її зміщення відносно початку файлу.

Давайте детальніше розберемося з його структурою:

uint32_t magic; MH_MAGIC_64 / MH_CIGAM_64

Magic - це так звані “магічні” числа які визначаються системою. Їх наявність пов’язана з тим історичним фактом що раніше різні комп’ютери

використовували різну послідовність зчитування байт з файлу. Магічні числа зберігають цю інформацію і кажуть про те що порядок байт відрізняється від стандартоного порядку байт системи на якій виконується код, це означає що всі байти мають бути розвернуті.

cpu_type_t cputype;

cpu_subtype_t cpusubtype;

Cputype та CpusubtypeВідповідно визначають тип процесора на якому буде виконуватись код можливі значення: i386, x86_64, arm64

uint32_t filetype;

FiletypeВідповідно визначає тип файлу, і може приймати велику кількість значень. Наприклад MH_EXECUTE визначає виконуваний файл, MH_DYLIB визначає динамічну бібліотеку MH_KEXT_BUNDLE системний драйвер, частина ядра.

uint32_t ncmds - Визначає кількість команд загрузки.

uint32_t sizeofcmds - Визначає розмір команд загрузки.

uint32_t flags - Додаткові флаги.

uint32_t reserved - Зарезервована частина для додаткового використання.

Розглянемо тепер команди загрузки. Це ті процедури які виконуються перед самим запуском виконуваного файлу. Серед всіх можна виділити ті які найчастіше присутні в файлі (рисунок 1.8):

- LC_SEGMENT - містить в собі різну інформацію про сегмент, такі як розмір, кількість секцій, зміщення в файлі до і після завантаження в пам'ять.
- LC_SYMTAB - завантажує в пам'ять таблицю символів та рядків.
- LC_DYSYMTAB - створює таблицю імпорту, дінні про символи беруться з таблиці символів.
- LC_LOAD_DYLIB - вказує залежність від сторонніх бібліотек.

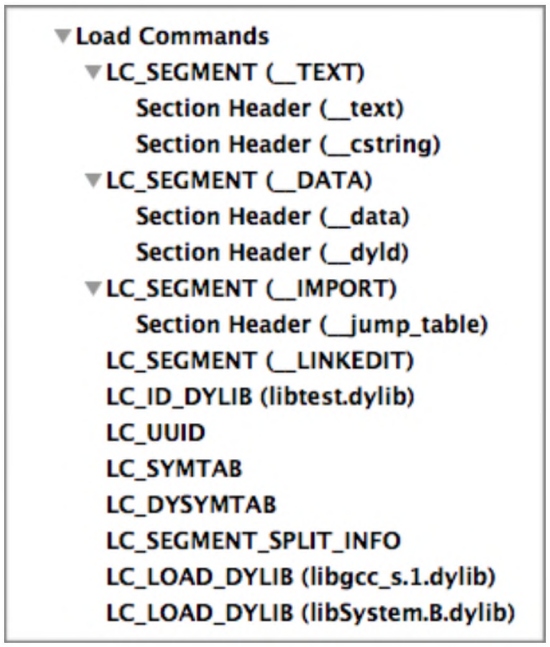


Рисунок 2.21 – Зображення команд загрузки виконуваного файлу

Будь яка команда загрузки має наступні компоненти:

uint32_t cmd - Команда в числовому вигляді.

uint32_t cmdsize - Розмір команди в байтах.

char segname[16] - Назва команди.

uint32_t vmaddr,

uint32_t vmsize - Адреса та розмір команди у віртуальній пам’яті.

Наступна частина структури Mach-O файлів - це сегменти. Сегменти можуть бути різного розміру і складу. В тілі файлу завжди наявні наступні важливі сегменти:

- __TEXT - в цьому сегменті знаходиться виконуваний код та інші данні які призначені тільки для зчитування.
- __DATA - данні які доступні для запису, в тому числі таблиці імпорту, які мають властивість змінюватись динамічним загружчиком під час кожного зв’язування.
- __OBJC - сегмент з різною інформацією стандартної бібліотеки мови Objective-C.
- __IMPORT - таблиця імпорту, варто зазначити, що вона є тільки в 32 бітних варіантах виконуваного файлу Mach-O.
- __LINKEDIT - в цьому сегменті динамічний загружчик розміщаєє свої данні для вже загружених сторонніх модулей: таблиці символів, рядків.

Найбільш цікавими секціями в перелічених сегментах є наступні, вони представляють всю логіку додатку:

- __TEXT,__text - код програми.
- __TEXT,__cstring - секція в якій записані константні рядки.
- __TEXT,__const - різні чисельні константи.
- __DATA,__data - ініціалізовані змінні, рядки та масиви.
- __DATA,__la_symbol_ptr - таблиця вказівників на імпортовані файли (64 бітна система).

- __DATA,__bss - не ініціалізовані статичні змінні.
- __IMPORT,__jump_table - місця викликів функцій з імпортованих бібліотек (32 бітна система).

Секція містить в собі наступні поля: ім’я сегмента, ім’я самої секції, зміщення в файлі і адреса в пам’яті по якій динамічний завантажувач її розмістив.

Висновки до розділу 2

В даному розділі детально розглянуто всі етапи роботи системи Gatekeepr, його роботу з різними застосунками. Описано роботу та систему налаштувань Gatekeeper. Досліджено будову бінарний файлів операційної системи MacOS, та інструменти їх дослідження, що допоможе при подальшій розробці системи захисту.

3 АНАЛІЗ НЕДОЛІКІВ СИСТЕМИ СЕРТИФІКАЦІЇ ТА РОЗРОБКА РІШЕННЯ ПРОБЛЕМ СЕРТИФІКАЦІЇ ЗАСТОСУНКІВ MACH-O

3.1 Дослідження недоліків мов програмування для MacOS та iOS

Всі нативні застосунки для платформ iOS та MacOS розробляються за допомогою мов програмування Objective-C або (починаючи з 2014 року) Swift. Це мови високого рівня які надають зручний інтерфес розробникам.Objective-C - компілюєма об'єктно-орієнтована мова програмування яка використовується корпорацією Apple, мова побудована на основі мови C та парадигм мови Smalltalk (об'єкти спілкуються між собою за допомогою відправки повідомлень).

Розроблена компанією Apple, використовується в основному у Mac OS X та GNUStep — середовищах, розроблених на основі стандарту OpenStep, та Cocoa — бібліотеки компонентів для розробки програм. Програму на Objective-C що не використовує цих бібліотек можна скомпілювати для будь-якої платформи, яку підтримує gcc компілятор з підтримкою Objective-C.

Objective-C є розширенням C і тому будь-яку програму на C можна скомпілювати компілятором Objective-C.

Одним з найважливіших плюсів, а також його мінусів являється Objective-C runtime - динамічне рішення яке дозволяє приймати остаточні рішення щодо виконання програми не на етапі компіляції, а на етапі виконання.

Swift -багатопарадигмовакомпільованамова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta. Безкоштовний посібник мови програмування Swift доступний для завантаження у магазині iBooks.

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує найкращі елементи мов C і Objective-C, тому синтаксис звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду.

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу швидкодію. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація.

Всі можливості мов Objective-C та Swift породжують проблеми із захистом таких додатків. Так як ці мови є мовами які компілються в проміжковий код, то вони залишать дуже багато інформації про вихідний код програми. Ця інформація і допомагає взломщикам точніше та детальніше розуміти програму.

Нажаль у виконуваних файлах залишається інформація про назви використаних класів, та назви методів цих класів. В секції “objc_classlist” бінарного файлу ми можемо знайти список всіх класів. Всі змінні які належать даному класу знаходяться в структурі “objc_list_header”, а доступні методи знаходяться в структурі “objc_method”.

‘Ще однією великою загрозою є те, що всі константні строки залишаються без перетворень в коді, а також можуть бути легко змінені або видалені.

Аналіз недоліків Objective-C Runtime

Objective-C замислювався як надбудова над мовою C, що додає до нього підтримку об'єктно-орієнтованої парадигми. Фактично, з точки зору синтаксису, Objective-C - це досить невеликий набір ключових слів і керуючих конструкцій над звичайним C. Саме Runtime, бібліотека часу виконання, надає той набір функцій, які роблять мову можливою, реалізуючи його динамічні можливості і забезпечуючи функціонування ООП .

Функції Runtime бібліотеки

Крім визначення основних структур мови, бібліотека включає в себе набір функцій, що працюють з цими структурами. Їх можна умовно розділити на кілька груп (призначення функцій, як правило, очевидно з їхньої назви):

- Маніпулювання класами: class_addMethod, class_addIvar, class_replaceMethod

- Створення нових класів: class_allocateClassPair, class_registerClassPair

- Інтроспекція: class_getName, class_getSuperclass, class_getInstanceVariable, class_getProperty, class_copyMethodList, class_copyIvarList, class_copyPropertyList

- Маніпулювання об'єктами: objc_msgSend, objc_getClass, object_copy

- Робота з асоціативними посиланнями

Звичайно Objective-C Runtime дуже зручний інструмент для розробки, але він дає багато можливостей для реверс-інженеру. За допомогою цієї особливості можна під’єднатися до виконуваного файлу і не тільки зчитувати данні, а й змінювати їх під час виконання програми, отримувати доступ до всіх важливих класів таким чином досліджуючи їх роботу.

3.2 Що таке Fat-Binary або Universal binary files

Універсальні бінарні файли - це виконувані файли або бандли які можуть працювати на процесорах різної архітетури. Наприклад PowerPC або Intel-32 або Intel-64. Ще відомі як FAT-binaries.

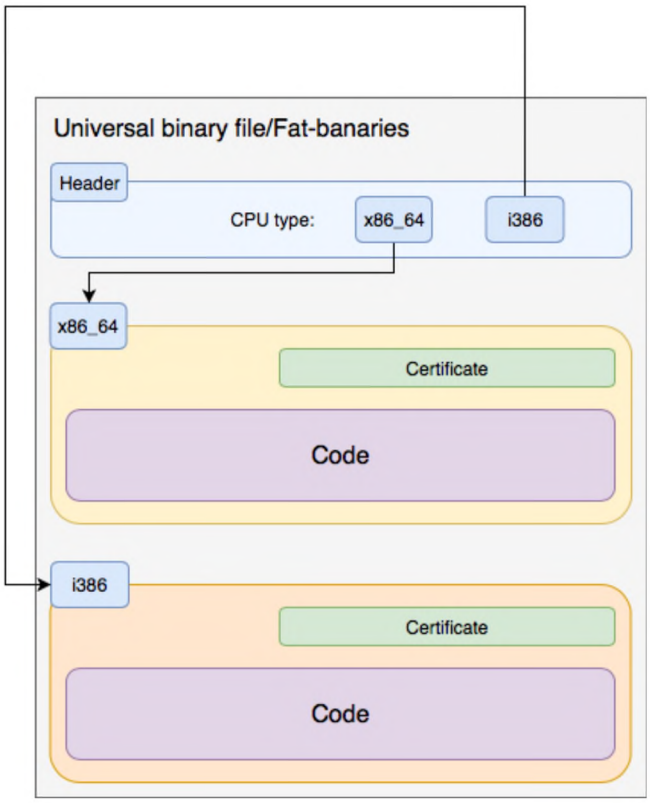


Рисунок 3.1 – Структура Fat-Binaries

Універсальний бінарний формат був представлений на конференції Apple WWDC 2005, як спосіб полегшити перехід від існуючої архітектури PowerPC до систем, що базуються на процесорах Intel, які почали використовуватись в 2006 році. В той час більшість виконуваних файлів включали в собі як версії для PowerPC, так і для x86. Операційна система виконуючи універсальний файл, шукала відповідний заголовок і виконувала відповідний розділ для використовуваної архітектури. Це дозволяло застосунку запускатися на будь-

якій архітектурі, без впливу на швидкість роботи, не враховуючи проблеми збільшення розміру виконуваних файлів.

Починаючи з версії 10.6 Mac OS X Snow Leopard, операційна система підтримує лише Intel архітектуру, тому система не вимагає використання Universal binaries при розповсюдженні застосунків. В даний час, universal binaries файли потрібні лише для забезпечення, призначеного для зворотної сумісності зі старими версіями Mac OS X, що працюють на старіших пристроях.

Причини такого рішення:

Існує два загальних альтернативних рішення. Перший спосіб полягає в тому, щоб просто компілювати два окремих двійкові файли, один зі скомпільованих для архітектури x86 і один для архітектури PowerPC. Однак це може ускладнити життя користувачу програмного забезпечення. Інша альтернатива полягає в тому, щоб в використовувати емуляцію однієї архітектури системою, яка працює на іншій архітектурі. Цей підхід призводить до зниження продуктивності і, як правило, розглядається як проміжне рішення, яке буде використовуватися тільки до тих пір, поки не будуть доступні інші рішення. Універсальні бінарні файли займають більше місця на жорсткому диску, оскільки потрібно зберігати кілька копій виконуваного коду.

3.3 Детальний опис вразливості перевірки сертифікатів застосунків для операційної системи MacOS

Знайдений обхід застосовуваного сторонніми розробниками API для підпису коду дозволяє представити будь-який код як підписаний Apple.

Подробиці про вразливість

Суть уразливості в неоднаковою перевірці підпису коду загрузчиком Mach-O і Code Signing API, які використовуються неправильно. Цю різницю можна експлуатувати за допомогою спеціально сформованого виконуваного файлу Universal / Fat.

Що таке файл Fat / Universal?

Fat / Universal - це двійковий формат, який містить кілька файлів Mach-O (виконуваний файл, dyld або пакет), кожен з яких орієнтований на певну архітектуру CPU (наприклад, i386, x86_64 або PPC).

Необхідні умови

Перший Mach-O в файлі Fat / Universal повинен бути підписаний Apple, це може бути файл i386, x86_64 або навіть PPC.

Самоподпісанний шкідливий бінарник або сторонній код повинен бути скомпільовано під i386 для macOS x86_64.

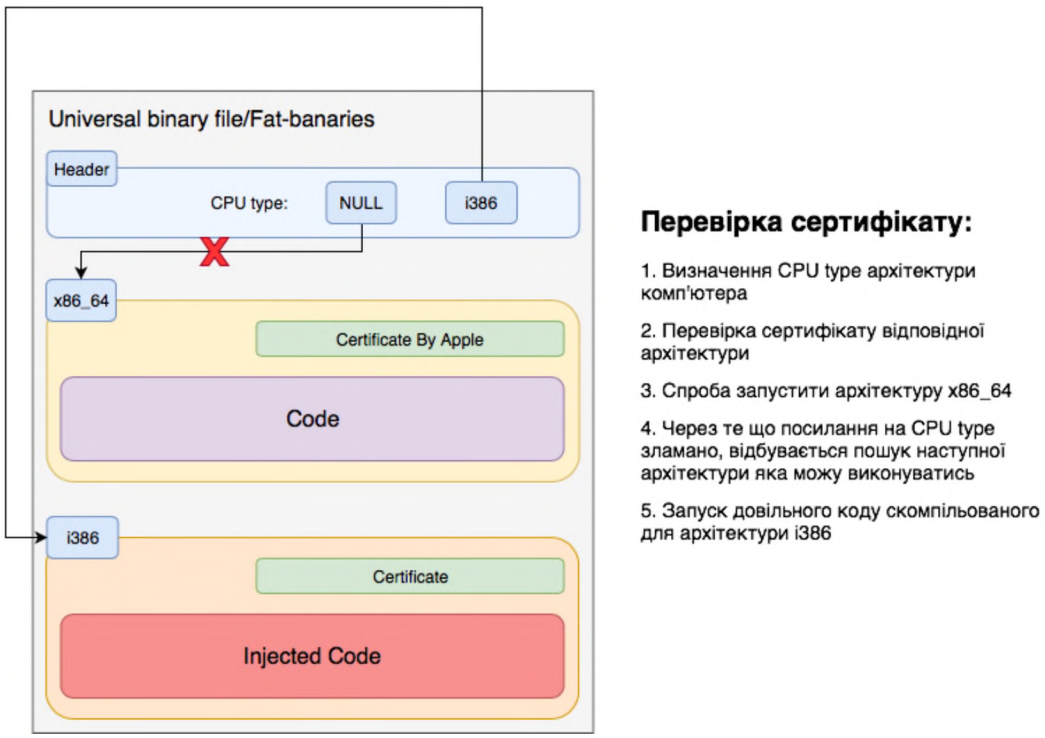


Рисунок 3.2 – Схема вразливості бінарних файлів

CPU_TYPE в заголовку Fat бінарники Apple повинен бути встановлений на неприпустимий тип або тип процесора, не рідний для чіпсета хоста

3.4 Спроба реалізації вразливості

Для того щоб реалізувати вразливисть системи перевірки codesign спочатку потрібно вибрати файл в який буде додано наш шкідливий код. Нехай це буде виконуваний файл Python, так як поставляється із системою за замовчуванням.

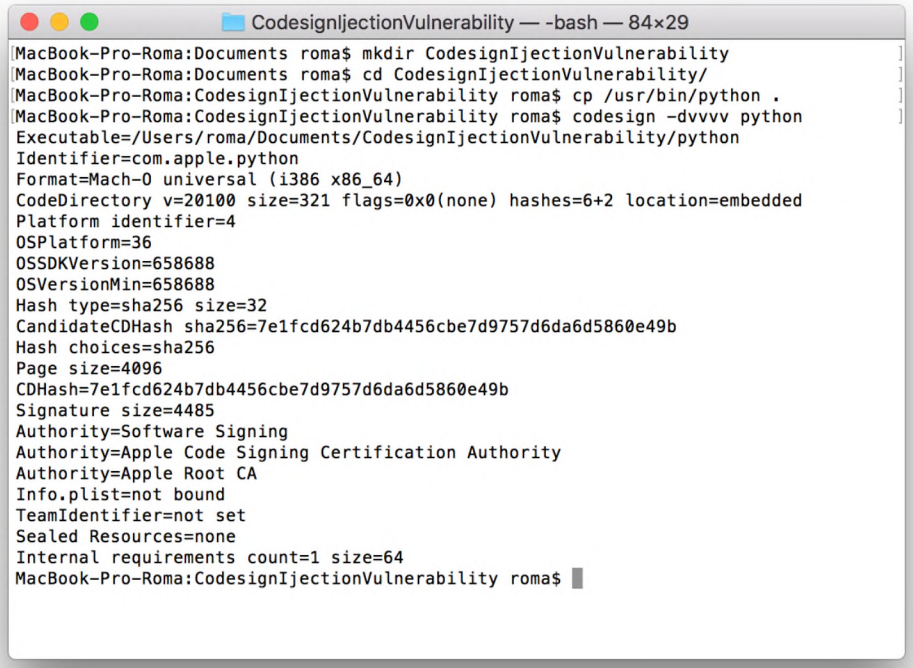


Рисунок 3.3 – Перевірка вбудованого файлу Python

Створимо тимчасову лакацію та скопіюємо python. За допомогою утиліти codesign можемо перевірити його цифровий підпис.

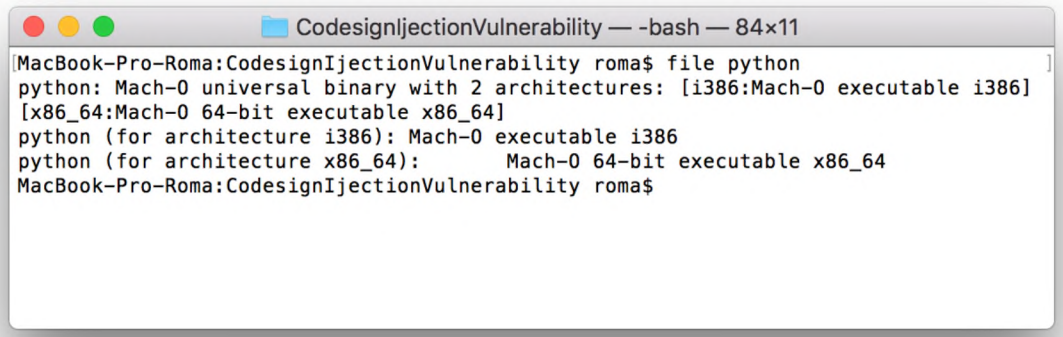


Рисунок 3.4 – Перевірка вбудованого файлу Python

Поле Authority вказує на те, що цей бінарний файл підписаний сертифікатом Apple, що ми в принципі очікували. Переглянемо якого типу цей бінарний файл за допомогою команди file.

Бачимо що це так званий Fat-Binary, тому що він містить в собі код для виконання для архітектури i368 та архітектури x86_64. Нам знадобиться тільки частина для архітектури x86_64, адже саме ця частина повинна перевірятися на відповідність до сертифікату.

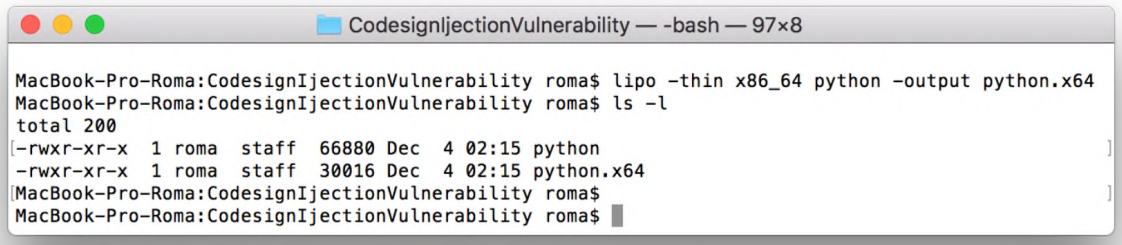


Рисунок 3.5 – Перевірка архітектур файлу Python

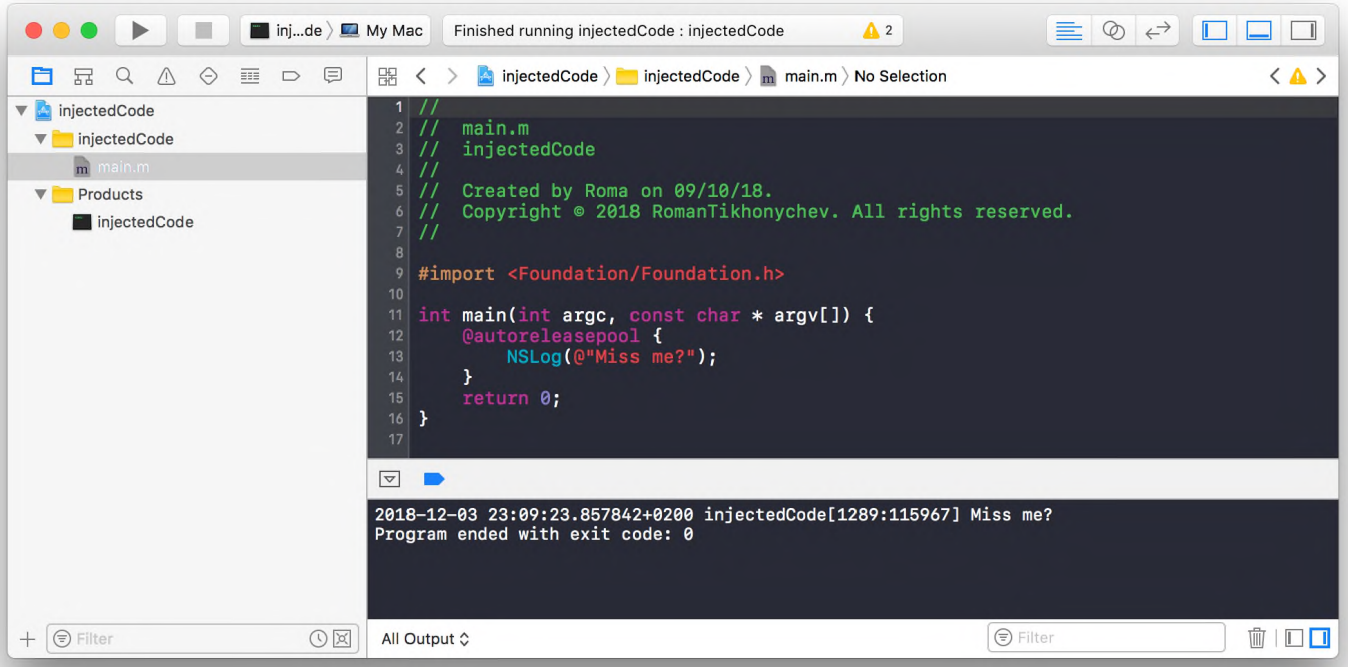


Рисунок 3.6 – Розробка Injectable коду

За допомогою утиліти lipo ми можемо роз'єднати fat-binary файл, і виділити ту архітектуру що нам потрібна

Тепер давайте скомпілюємо бінарний файл, який буде містити наш шкідливий код. Для цього створюємо новий проект в Xcode. Поки нам неважливо який саме код буде містити наш виконуваний файл. Додамо в нього вивід строки, щоб пізніше було простіше дізнаватися код для якої архітектри буде виконуватись.

Так як один fat-binary файл не маже містити в собі однакові архітектури, а код все-одно повинен виконуватись на сучасних процесорах Intel, в налаштуваннях проекту потрібно виставити флаги

VALID_ARCHS = i386

ARCHS = $(ARCHS_STANDARD_32_BIT)

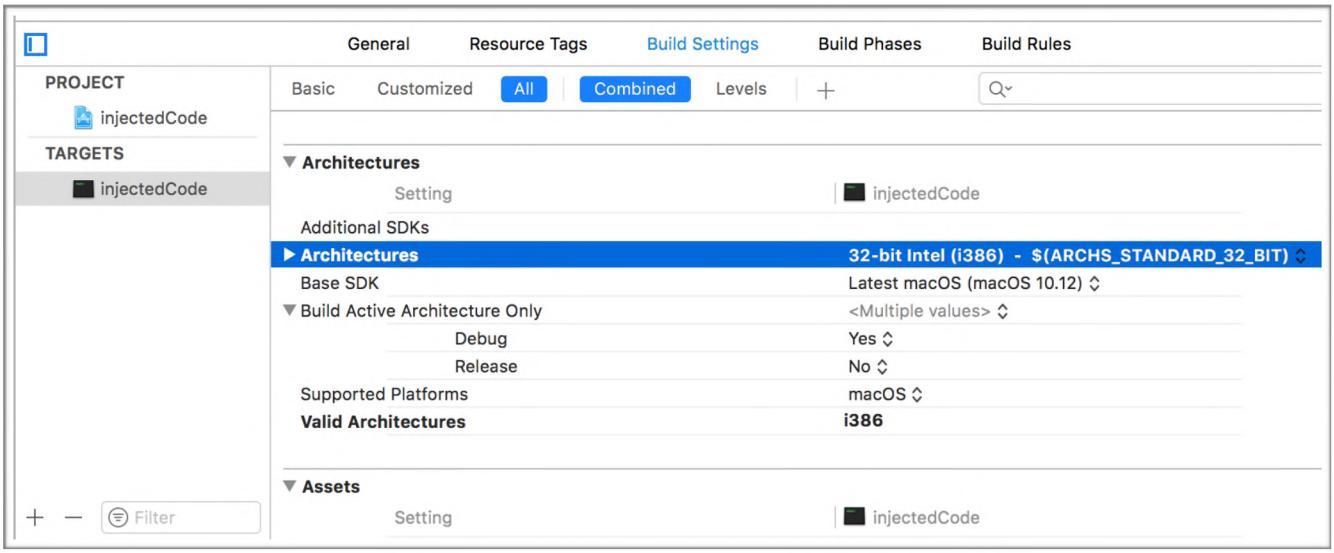


Рисунок 3.7 – Налаштування архітектури Fat-Binaries

Тепер скопмілюємо наш виконуваний файл, та перевіримо його за допомогою комнди file, повинні отримати файл який буде скомпільовани для архітектури i368

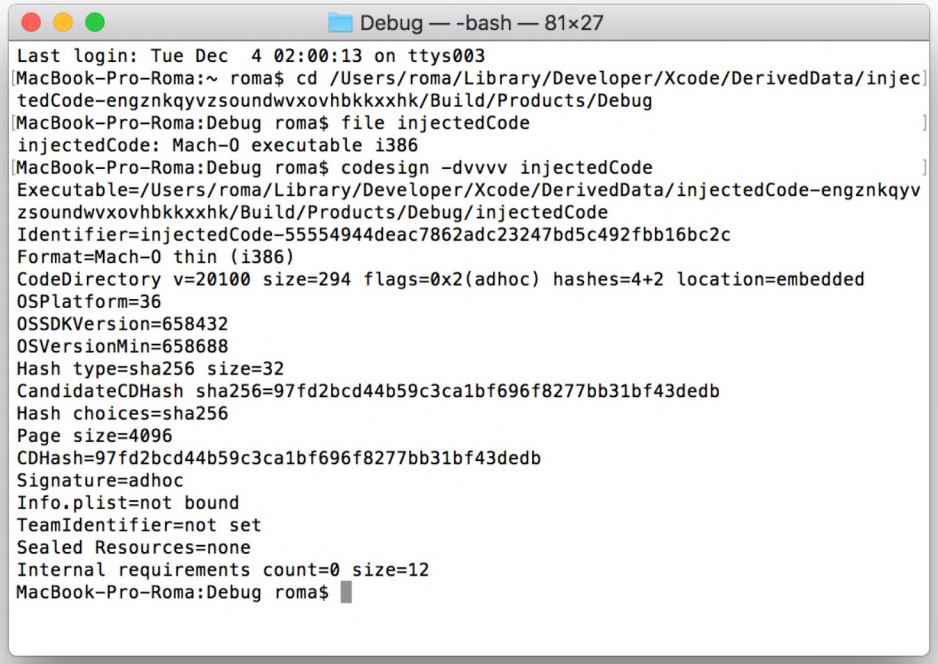


Рисунок 3.8 – Перевірка архітектур файлу Python

Важливим є те, що Xcode сам підпише його в режимі AdHoc, про це свідчить поле Signature. Це означає що цифровий підпис було додано до бінарного файлу, в даному випадку він був підписаний моїм Apple Developer ID. Якщо перевірити його через Gatekeeper то отримаємо reject

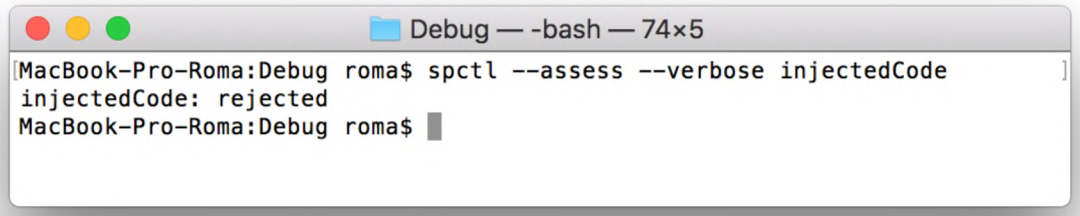


Рисунок 3.9 – Перевірка архітектур файлу Python

За допомогою команди lipo -create, згенеруємо наш шкідливий fat-binary файл, який об’єднає наш injectedCode та python.x64 в один виконуваний файл

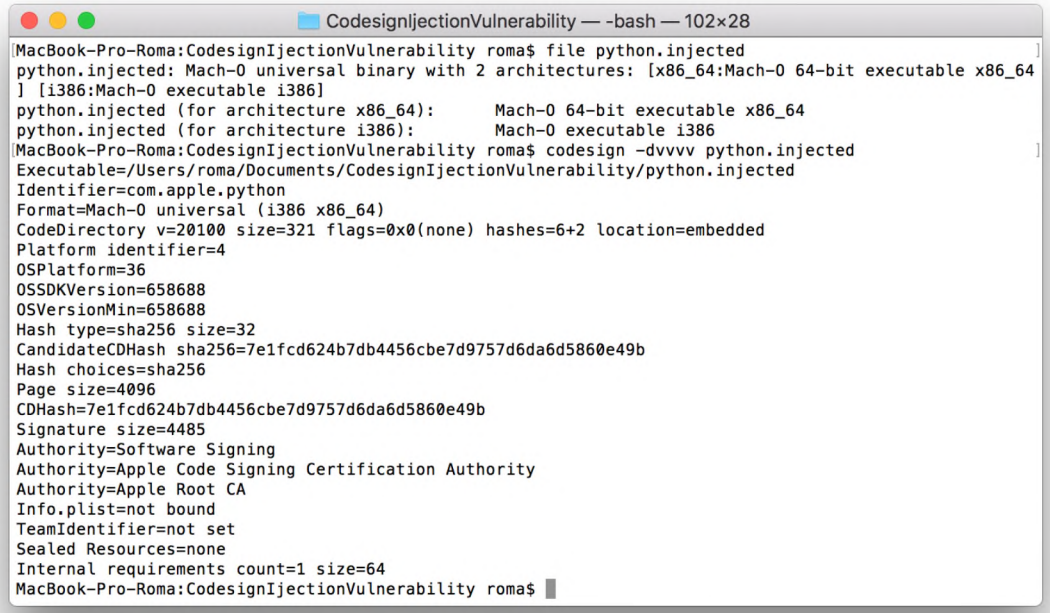


Рисунок 3.10 – Перевірка архітектур файлу Python

Отримали файл, з двома архітектурами. Саме тут з’являється вразливість такого виконуваного файлу. Якщо ми перевіремо його підпис, він перевіриться тільки для однієї архітектури, а саме для архітектури яка підходить для виконання на даній системі - x86_64

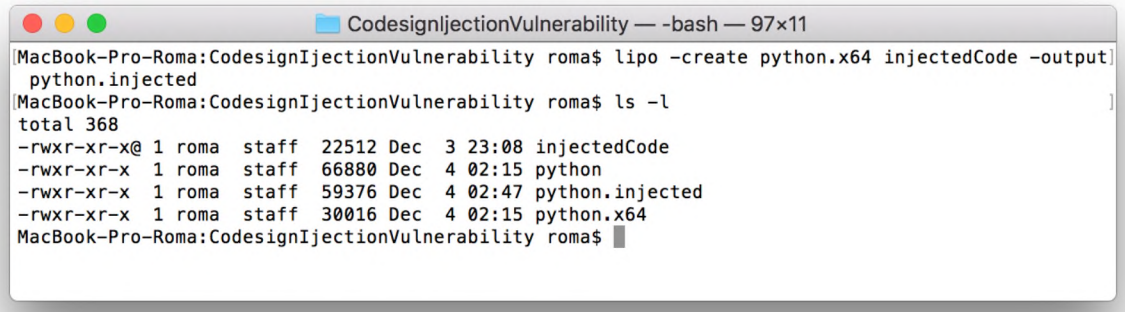


Рисунок 3.11 – Перевірка архітектур файлу Python

Але якщо ми запустимо виконувани файл, то виконається оригінальни код python. Щоб змінити це, нам потрібно зламати параметр CPU_TYPE та CPU_SubType для архітектури x86_64. Використовуючи утиліту Mach-O-View можна легко та швидко змінити CPU_TYPE та CPU_SubType.

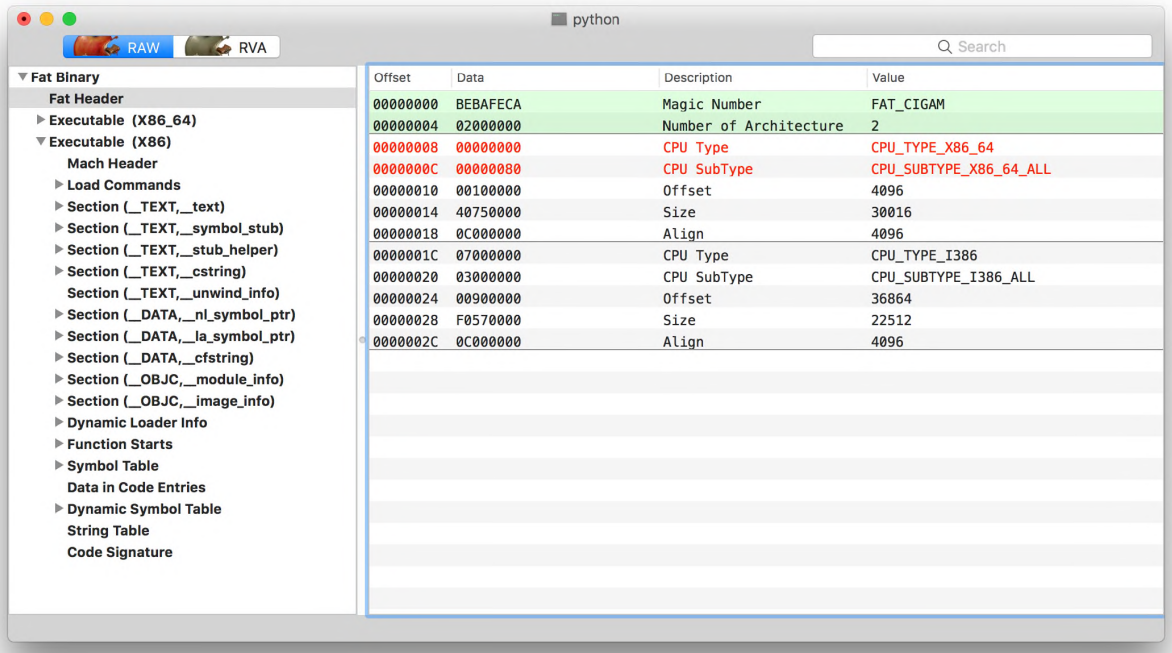


Рисунок 3.12 – Перегляд архітектур в програмі MacOS

Після цієї операції зберігаємо python файл, та можемо сміливо його запускати. В результаті отримуємо виконання нашого шкідливого коду, при цьому перевірка сертифікату дає хибний результат

Authority=Apple Code Signing Certification Authority

Authority=Apple Root CA

Таким чином ми можем отримати виконуваний файл, який проходить перевірки підпису та не перевіряється системою Gatekeeper але насправді являється шкідливим програмним забезпеченням.

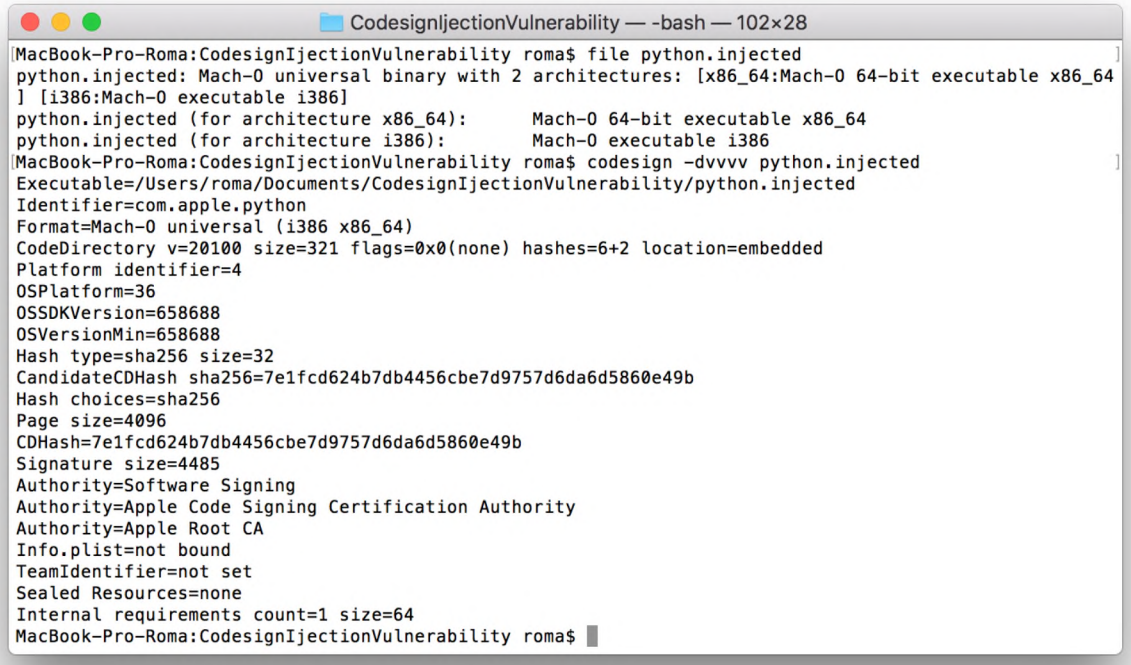


Рисунок 3.13 – Перевірка архітектур файлу Python

Перевірка отриманого файлу через відомі утиліти та антивіруси MacOS

Ми отримали потенційно шкідливий виконуваний файл python, в якому можемо розмістити будь який код. Сдантдартна утиліта codesign визначає його як файл, який було підписано сертифікатом Apple. Для того щоб зрозуміти чи вже є готові рішення проблеми перевірки сертифікатів, порівняємо перевірки цього файлу, в різних популярних утилітах.

VirusTotal - є безкоштовним сервісом для перевірки файлів на наявність в них вірусів, та шкідливого коду. Він використовує власну накоплену базу шкідливих файлів, а також перевіряє наданий файл через базу із 57 антивірусів

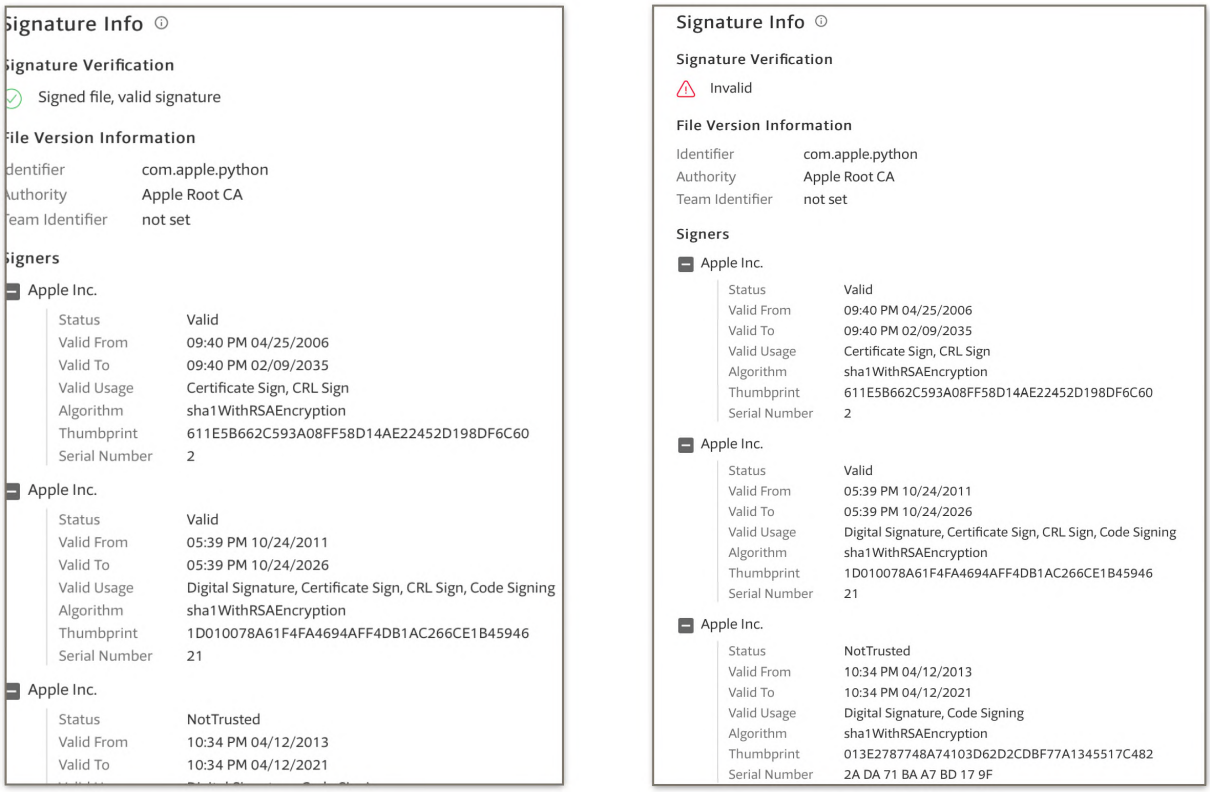


Рисунок 3.14 – Перевірка шкідливого та не шкідливого файлу через VirusTotal

серед яких є найпопулярніші Avast, BitDefender, DrWeb, Kaspersky та багато інших. Результат перевірки зображені на рисунку

Бачимо що сервіс VirusTotal все-таки правильно визначив те, що підпис не валідний. Але поле Signers залишилось не змінним, і все ще вказує на те, що бінарний файл було підписано компанією Apple.

Утиліта RB App Checker lite - безкоштовна утиліта доступна через магазин MacAppStore яка показує підпис виконуваного файлу. Результати перевірки є ідентичні з сервісом VirusTotal

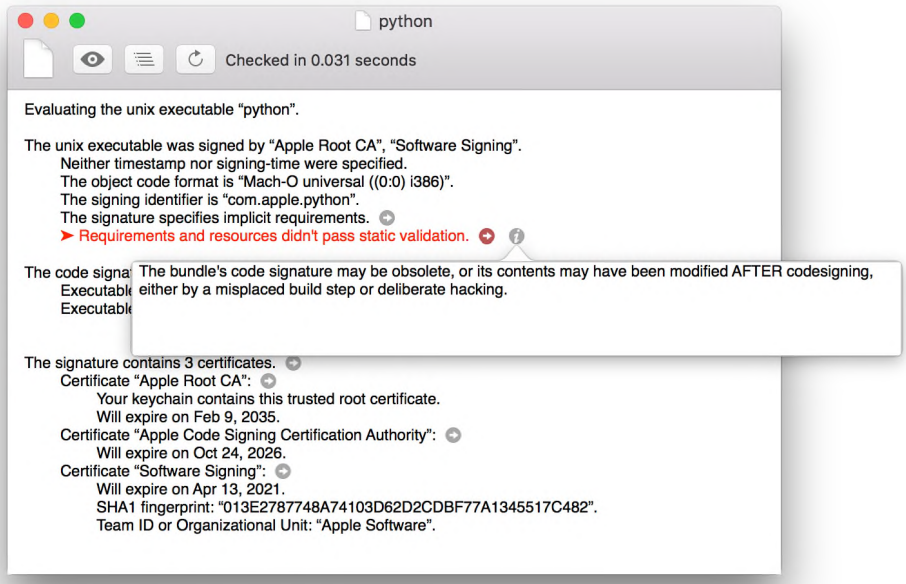


Рисунок 3.15 – Перевірка фалу через утиліту RB Checker

Утиліта WhatsYourSign - невеликий інструмент який в зручномій для користувача формі показує підпис та валідність виконуваних файлів.

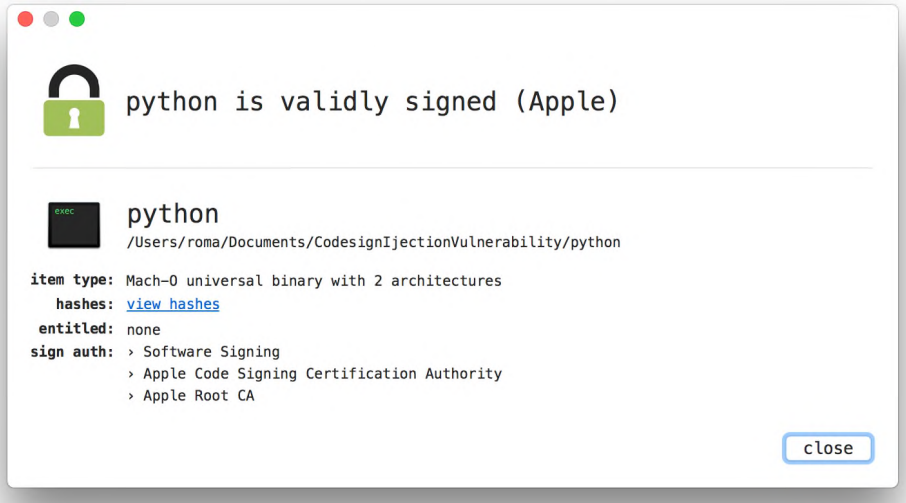


Рисунок 3.16 – Перевірка фалу через утиліту WhatsYourSign

Результат перевірки показує що дана утиліта не передбачає правильної перевірки Fat-binaries файлів, а це означає що користувачі які полягаються на її використання потенційні жертви такої вразливості.

3.5 Можливі рішення проблеми верифікації підпису

Як правило, розробники перевіряють бінарники Mach-O або Fat / Universal за допомогою API SecStaticCodeCheckValidityWithErrors () або SecStaticCodeCheckValidity () з наступними прапорами:

kSecCSStrictValidate

KSecCSCheckAllArchitectures

Ці прапори повинні гарантувати, що весь завантажений в пам'ять код у файлі Mach-O або Fat / Universal криптографически підписаний. Проте, ці API за замовчуванням не забезпечують належну перевірку, так що стороннім розробникам потрібно виокремлювати кожну архітектуру в файлі Fat / Universal і перевіряти, що identities збігаються і криптографически надійні.

Кращий спосіб перевірити кожну вкладену архітектуру в файлі Fat / Universal - це спочатку викликати SecRequirementCreateWithString з вимогою "anchor apple", потім SecStaticCodeCheckValidity з прапорами kSecCSDefaultFlags | kSecCSCheckNestedCode | kSecCSCheckAllArchitectures | kSecCSEnforceRevocationChecks з посиланням на вимогу; як показано в пропатченний вихідному кодеWhatsYourSign.

Передаючи "anchor apple", в функцію SecRequirmentCreateWithString, такий виклик діє аналогічно команді codesign -vv -R = 'anchor apple', вимагаючи наявності ланцюжка довіри Apple Software Signing для всіх вкладених бінарників в файлі Fat / Universal. Крім того, передаючи прапори і вимога до SecStaticCodeCheckValidity, все архітектури перевіряються на цю вимогу, і застосовуються перевірки відкликання.

3.6 Проектування та розробка власного застосунку

Проектування власного застосунку для перевірки сертифікатів застосунків MacOS

Файли Kext це драйвери для macOS. Слово "Kext" скорочення від Kernel Extensions - це розширення ядра macOS. Коли ви завантажуєте вашу машину, код, що міститься в цих кексах, автоматично вводиться в операційну систему.

Kext написано на об'єктно-орієнтованим методом за допомогою підмножини мови C ++. Механізм kext працює у вигляді плагінів з можливістю загрузки під час виконання коду ядра, це означає, що ви можете вручну або програмно завантажувати або вивантажувати частини коду без необхідності перезапуску або завершення роботи комп'ютера. Kext може обробляти данні з апаратного забезпечення (відеокарти, камери, аудіосистеми, центральний процесор тощо)

Найкращим рішенням буде написати драйвер для операційної системи, який на старті будь-якого додадку буде перевіряти надійність цифрового підпису.

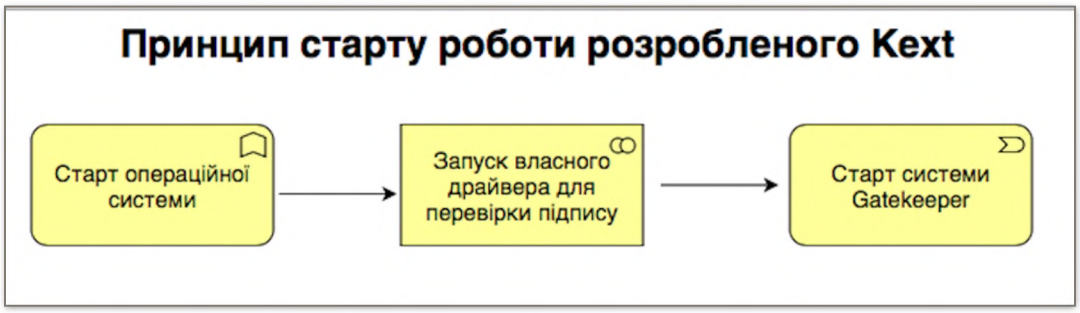


Рисунок 3.17 – Принцип старту роботи розробленого Kext

Розробка коду власного застосунку для перевірки сертифікатів застосунків MacOS

Як ми визначили раніше найкраащий спосіб перевірити кожну вкладену архітектуру в файлі Fat / Universal - це спочатку викликати SecRequirementCreateWithString з вимогою "anchor apple", потім SecStaticCodeCheckValidity з прапорами kSecCSDefaultFlags | kSecCSCheckNestedCode | kSecCSCheckAllArchitectures | kSecCSEnforceRevocationChecks. Враховуючи ці умови функція перевірки сертифікатів буде виглядати наступним чином:

```
//determine if a file is signed by Apple proper
BOOL isApple(NSString* path, SecCSFlags flags)
{
    //flag
    BOOL isApple = NO;

    //code
    SecStaticCodeRef staticCode = NULL;

    //signing reqs
    SecRequirementRef requirementRef = NULL;
```

```
    //status
    OSStatus status = -1;

    //create static code
    status = SecStaticCodeCreateWithPath((__bridge CFURLRef)([NSURL
fileURLWithPath:path]), kSecCSDefaultFlags, &staticCode);
    if(noErr != status)
    {
        //bail
        goto bail;
    }
    //create req string w/ 'anchor apple'
    // (3rd party: 'anchor apple generic')
    status = SecRequirementCreateWithString(CFSTR("anchor apple"),
kSecCSDefaultFlags, &requirementRef);
    if( (noErr != status) ||
        (requirementRef == NULL) )
    {
        //bail
        goto bail;
    }
    //check if file is signed by apple by checking if it conforms to req string
    // note: ignore 'errSecCSBadResource' as lots of signed apple files return this
issue :/
    status = SecStaticCodeCheckValidity(staticCode, flags, requirementRef);
    if( (noErr != status) &&
        (errSecCSBadResource != status) )
    {
        //bail
```

```
        // just means isn't signed by apple
        goto bail;
    }
    //ok, happy (SecStaticCodeCheckValidity() didn't fail)
    // file is signed by Apple
    isApple = YES;
bail:
    //free req reference
    if(NULL != requirementRef)
    {
        //free
        CFRelease(requirementRef);

        //unset
        requirementRef = NULL;
    }

    //free static code
    if(NULL != staticCode)
    {
        //free
        CFRelease(staticCode);

        //unset
        staticCode = NULL;
    }

    return isApple;
}
```

Давайте перевіримо працездатність даного коду, перевіримо використані раніше python бінарні файли.

Виконавши перевірку оригінального файлу отримуємо результат:

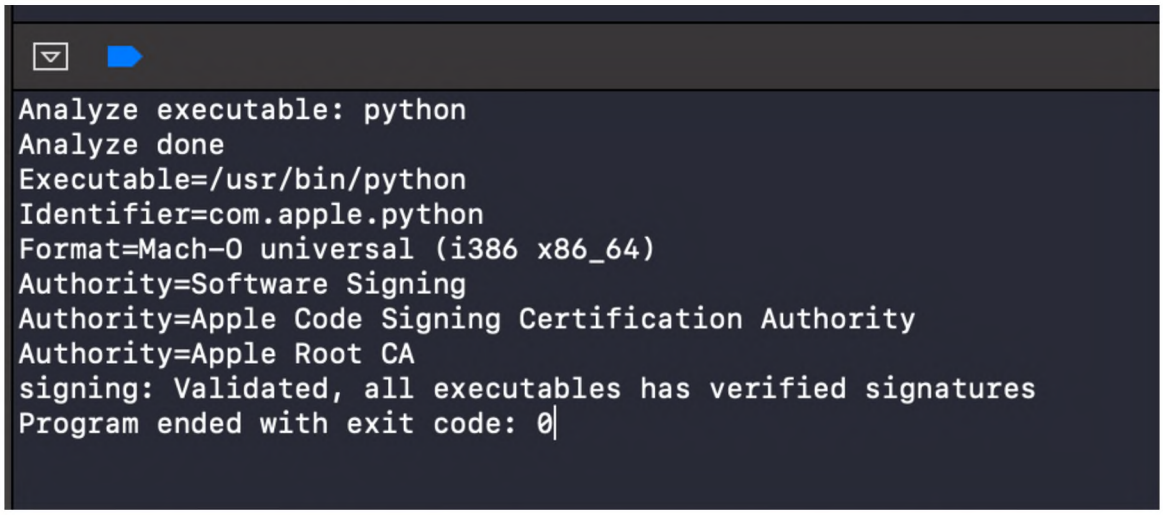


Рисунок 3.18 – Перевірка виконуваного файлу Python

Тоді як перевірка шкідливого файле pythonInjectable видає результат:

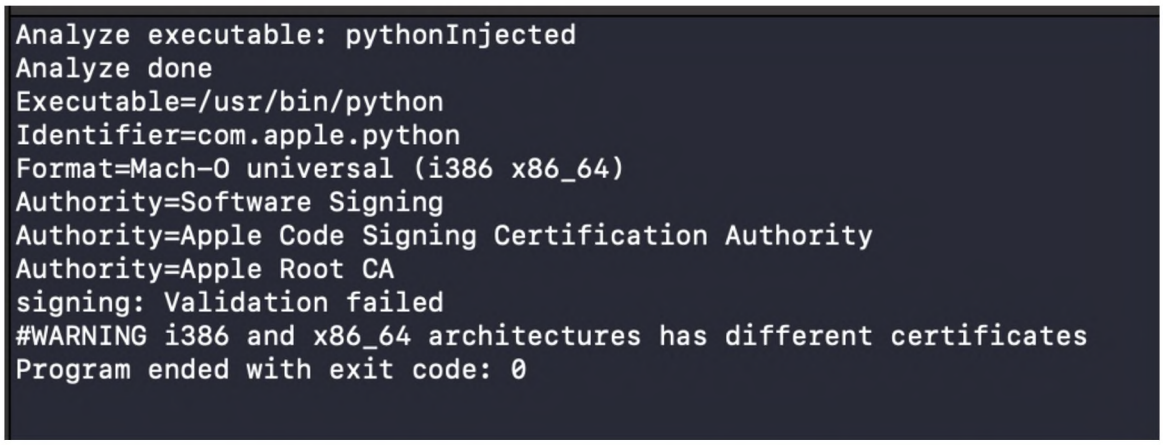


Рисунок 3.19 – Перевірка виконуваного файлу PythonInjectable

В результаті перевірки засвідчуємось, що наш код правильно визначає сертифікати, якими було підписано кожну із архітектур бінарного файлу. В

результаті перевірки підпису визначає що файл було зламано і його сертифікати не відповідають дійсності.

Висновки до розділу 3

В ході роботи було проаналізовано вразливість системи перевірки сертифікатів застосунків MacOS. Виконана спроба реалізації вразливості системи перевірки підпису застосунків MacOS. Зпроектовано та розроблено систему захисту для вирішення проблеми вразливості цієї перевірки. Запропоновану систему було протестовано та порівняно вже з існуючими інструментами. Результати тестування показали успішні результати, що доводить правильність розробленої системи захисту.

4 РОЗРОБКА СТАРТАП ПРОЕКТУ

Звичайно кожна розробка може містити як наукову, так і комерційну складову. В наступному розділі я хотів би оцінити доцільність розробки комерційного проекту або стартапу на основі запропонованих рішень. В результаті сформувати кроки формування комерційного проекту.

4.1 Опис ідеї проекту

Для початку треба чітко сформувати основні положення ідеї. Проаналізувати її можливі сфери використання, користувача та його проблеми. Провести пошук конкурентів та зробити порівняння з вже існуючими рішеннями. Оцінити конкурентність ринку.

Таблиця 4.1 - Опис ідеї стартап-проекту

Зміст ідеї	*Напрямки застосування*	*Вигоди для користувача*
Розробка застосунку для перевірки вдосконалення вбудованої системи перевірки сертифікації застосунків MacOS. Постачання і впровадження засобів захисту.	1. Використання продукту компаніями середнього і великого розміру, для підвищення загального рівня безпеки	Безпека внутрішньої мережі. Використання вдосконаленої системи перевірки сертифікатів застосунків
	2. Використання окремими користувачами	Використання цілісної та вдосконаленої системи перевірки та захисту системи MacOS
	3. Використання застосунку в державних системах та мережах	Можливість контролю механізму перевірки сертифікатів. Покращення системи безпеки всієї мережі

Проаналізувавши схожих конкурентів

Таблиця 4.2 - Визначення сильних, слабких та нейтральних характеристик ідеї проекту

2	*3*		*4*	*6*
Техніко-економічні характеристики ідеї	*(потенційні) товари/концепції конкурентів*		*W (слабка сторона)*	*S (сильна сторона)*
	Мій проект	*RB App Checker*		
економічні	Приблизні витрати на утримання офісу та співробітників. Проектування та розробку проекту - 10000$	Приблизні витрати на утримання офісу та співробітників. Проектування та розробку проекту - 20000$	Недостатній рівень витрат на маркетинг	Значно дешевша реалізація проекту. Використання нової бази коду. Використання повного циклу перевірки бінарних файлів.

Продовження таблиці 4.2.

2	3		7
технічні	Використання API для первірки підпису застосунків в SecStaticCodeCheckValidity	Перевірка лише верхнього рівня сертифікатів, не всього бінарного файлу	Можливість використання системи на всіх актуальних версіях операційних системах
надійності	Використання API для первірки підпису застосунків в SecStaticCodeCheckValidity	Нажаль даний продукт не реалізовує повну перевірку, тому не можна вважати його надійним	Досягнення вищого рівня перевірки. Виконання перевірки на рівні системного драйвера.
технологічні	Використання платформ, що надають інформацію про загрози, індикатори компрометації, оновлення та виправлення для програмного забезпечення, загальний стан інформаційної безпеки	Відсутність таких компонентів	Отримання більшої кількості інформації для підвищення ефективності роботи SOC та якості надаваних послуг

естетичні	Зручність та простота використання (інтегровано з системою)	Для використання потрібно додатково перевіряти виконуваний файл перед запуском	Можливість налаштуваннясистеми для окремого користувача

Продовження таблиці 4.2.

2	3		4	6
органолептичні	Швидкість роботи рішення для користувача базується на тісній інтеграції з системою	Не є швидким рішенням з точки зору використання	Розробка системи в форматі kext дає можливість якнайкраще інтегрувати її з системою	Просте встановлення системи на комп'ютер користувача, не потребує додаткових налаштувань
транспортабельності	Має максимальну простоту для розповсюдження	Встановлення потреує доступ до інтернеті, можливе личе через офіційний магазин AppStore	Недостатній рівень контролю за розповсюдженням його без лецензій	Загрузка драйвера з системою дає найбільшу інтеграцію з продуктом

4.2 Технічний огляд ідеї проекту

Потрібно проаналізувати технології які потрібні для реалізації проекту. Висновком дослідження є те, що технології доступні та мають повний набір інструментів для реалізації проекту. Основні фінансові затрати буде витрачено на маркетингову складову, та розвиток популярності продукту на ринку. Результати дослідження описано в таблиці 4.4

Таблиця 4.4. – Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Реалізаціявласного драйвера системи	Мова програмування Objective-C / Swift	Впровадження	Доступна
2	Використані API	Готове рішення	Впровадження	Доступна
3	Розслідування інцидентів та аудит на базі використання системи логування	Мова програмування Objective-C / Swift	Впровадження	Доступна
4	Канали дистрибуції застосунку	Використання сторонніх сервісів для розповсюдження		
Обрана система реалізації проекту є відкритою та доступною на всіх актуальних системах MacOS та OS X. Всі виористані технологіє є доступними, тому й були використани при розробці				

4.3 Аналіз можливостей ринку для запуску стартап-проекту

Потрібно визначити можливості ринку, та оцінити які є можливості для впровадження проекту. Це дослідження допоможе скласти план та напрямки розвитку продукту. Важливо враховувати стан ринку, оцінити потенційного клієнта продукту, пропозиції конкурентів. Дослідження сформовані в таблиці 4.4

Таблиця 4.4 - Попередня характеристика потенційного ринку стартап-проекту

1	*2*	*3*
№ п/п	*Показники стану ринку хмарних SOC*	*Характеристика*
1	Кількість головних гравців, од	5+
2	Загальний обсяг продаж, грн/ум.од	> 50 млн ум.од
3	Динаміка ринку (якісна оцінка)	Помірно зростає
4	Наявність обмежень для входу	Великі початкові вкладення в маркетингову складову проекту
5	Специфічні вимоги до стандартизації та сертифікації	Відповідність стандартам розробки застосунків MacOS
6	Середня норма рентабельності в галузі, %	Приблизно 120%

За попередніми оцінками, ринок розробки систем безпеки для MacOS є привабливим, але вимагає для початку наявність стартового капіталу.

Час існування ринку можна оцінювати приблизно в 40 років, від початку розвитку ринку комп'ютерів Macintosh. Ринок стрімко розвивається і збільшується. З кожним роком кількість користувачів збільшується а значить що програмні рішення стають все більш популярнішими.

Користувач та його види, представлені в таблиці. 4.5

Таблиця 4.5 - Характеристика потенційних клієнтів стартап-проекту

№ п/п	*Потреба, що формує ринок*	*Цільова аудиторія (цільові сегменти ринку)*	*Відмінності у поведінці різних потенційних цільових груп клієнтів*	*Вимоги споживачів до товару*

1	Безпека запуску застосунків. Перевірка достовірності сертифікатів застосунків. Моніторинг безпеки інформаційних систем	Окремі користувачі Невеликі корпоративні компанії Державні компанії та підприємства	Для кожної групи користувачав важливі різні параметри системи. Так для окремик користувачів важливо зручність та швидкість використання даної системи. Простота налаштування та роботи. Для невеликих компаній важливо мати механізм швидкого встановлення на велику кількість машин. Можливість відслідковування та аудиту за допомогою системи логування. Для державних компаній важливо відповідати всім нормам розробки, та отримати всі сертифікати відповідності для застосування у державному секторі. Але варто зазначити що для всіх груп користувачів найважливішим є достовірність перевірки та роботи механізму	Швидкодія роботи. Повний аналіз сертифікатів виконуваних застосунків. Наявність системи логування для відслідковування підозрілої активності. Відповідність до сертифікатів розробки.

Таблиця 4.6 - Фактори загроз

№ п/п	*Фактор*	*Зміст загрози*	*Можлива реакція компанії*
1	Цінова конкуренція	Коливання цін на послуги конкурентів	Пошук шляхів зниження вартості послуг
2	Зниження доходів потенційних споживачів	Зниження купівельної спроможності клієнтів	Вимушене зменшення обсягів виробництва
3	Збільшення розміру податків	Відтік коштів із сфери виробництва до бюджету	Пошук шляхів мінімізації податків
4	Рівень інфляції	Знецінювання коштів, ріст різниці в курсах валют	Отримання довгострокового кредиту

Таблиця 4.7 - Фактори можливостей

№ п/п	*Фактор*	*Зміст можливості*	*Можлива реакція компанії*
1	Поява нових технологій та високоефективного обладнання	Розширення спектру надаваних послуг для клієнтів, збільшення кількості клієнтів, підвищення ефективності роботи сервісів, зменшення витрат на роботу сервісів	Швидке впровадження нових технологій завдяки хмарним сервісам, підвищення вартості послуг, створення нових сервісів для клієнтів, проведення маркетингової кампанії
2	Стабілізація політичного та економічного становища в державі	Зменшення рівня інфляції, збільшення кількості клієнтів	Спроби лобіювання інтересів компанії в держаних установах
3	Збільшення кількості кібер-атак, діяльності зловмисного програмного коду тощо	Збільшення попиту на послуги SOC	Залучення нових клієнтів, можливе підвищення вартості послуг

4	Залучення нових відомих компаній у якості клієнтів чи постачальників	Збільшення впливу бренду Cloud SOC на ринку послуг кібербезпеки, зменшення цін на рекламу та маркетинг	Розробка та впровадження нових засобів та програмних комплексів для розширення спектру послуг у сфері кібербезпеки

Таблиця 4.8 - Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	*В чому проявляється дана характеристика*	*Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)*
1. олігополістична конкуренція	Динаміка цін, яка майже не залежить від рівню попиту на продукцію; конкуренція зміщуються в площину реклами, рівня якості продукції та індивідуалізації	Вдосконалення рішення для підвищення якості надаваних послуг для клієнтів
2. За рівнем конкурентної боротьби - національний	Надання послуг для різних груп та типів клієнтів в Україні та за її межами	Застосування хмарних технологій для підвищення конкурентоспроможності, застосування нових технологій та підходів для вдосконалення рішення
3. За галузевою ознакою - міжгалузева	Рішення може використовуватися користувачами з різних галузей	Розширення сфери надання послуг, додання нових функцій сервісу та послуг для клієнтів
4. Конкуренція за видами товарів: - товарно-видова	Рішення, що використовується для задоволення потреб клієнтів, але істотно відрізняються від рішень конкурентів на користь якості сервісів	Надання послуг із розслідування інцидентів, аудиту та консалтингу в сфері інформаційної безпеки
5. За характером конкурентних переваг - цінова	Цінова	Надання ширшого спектру послуг, порівняно із конкурентами
6. За інтенсивністю - марочна	Сукупність характеристик та властивостей рішення	Підвищення якості роботи сервісів для клієнтів

Таблиця 4.9 - Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	*Потенційні конкуренти*	*Постачальники*	*Клієнти*	*Товари-замінники*
Складові аналізу	RB Checker App Objective-See Company Gatekeeper team	Розмір капіталовкладень; доступ до ресурсів у конкурентів; наявність патентів та товарних знаків.	Значення розмірів поставок для постачальників. Диференціація витрат.	Розміри закупівель державних підприємств та органів влади; рівень чутливості до зміни ціни; контроль якості	Ціна товару та змінні витрати
Висновки:	Наявна досить інтенсивна конкурентна боротьба з боку прямих конкурентів	Є можливості входу на ринок, але існує чимало серйозних конкурентів, що вже працюють на цьому ринку.	Постачальники диктують умови роботи на ринку; компанія, яка здійснює більшу кількість продажів, отримує привілеї та більші розміри знижок на товари та послуги	Клієнти диктують умови на ринку; при організації закупівель товарів та послуг, відчутний рівень відношення до вартості рішення та його якості.	Обмежень на ринку через товари замінники немає.

Запропонований проект, має важливі переваги серед конкурентів. До однозначно сильних сторін можна зазначити використаня зручного формату виконуваного коду, а саме драйвера для операційної системи MacOS. Він є зручнішим чим рішення запропоновані конкурентами. Підвищений інтерес можливих клієнтів вказує на те, що ця технологія буде актуальною на ринку, та буде постійно вдосконалюватись.

Таблиця 4.10 - Обґрунтування факторів конкурентоспроможності

№ п/п	*Фактор конкурентоспроможності*	*Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)*
1	Ціна та змінні витрати	Ціноутворення, яке не є однаковим на послуги для різних типів клієнтів (це обумовлено рівнем видатків за використання хмарних технологій для розміщення потужностей SOC).
2	Розміри закупівель замовників	SOC, який реалізований у хмарному середовищі, здатний легко масштабуватись у відповідності до потреб замовників послуг; це значно підвищує рентабельність проекту, порівняно з конкурентами, у яких потужності не розміщені «хмарі».
3	Доступ до ресурсів у конкурентів	Так як компанії-конкуренти вже працюють на ринку, вони мають клієнтів та встановлений шлях продажів та маркетингу, тому вони мають більше ресурсів як матеріальних, так і інформаційних
4	Гнучкість цін на послуги	На відміну від конкурентів, при застосуванні хмарних технологій, є можливість зменшення цін на послуги для користувачів
5	Рівень концентрації послуг	Спектр послуг, які можуть бути надані клієнтам є значно ширшим, в порівнянні з конкурентами

Таблиця 4.11 - Порівняльний аналіз сильних та слабких сторін

№ п/п	*Фактор конкурентоспроможності*	*Бали 1-20*	*Рейтинг товарів-конкурентів у порівнянні з запропонованим рішенням*						
			–3	*–2*	*–1*	*0*	*+1*	*+2*	*+3*
1	Ціна та змінні витрати	18			+				
2	Розміри закупівель замовників	10	+						
3	Доступ до ресурсів у конкурентів	4							+
4	Гнучкість цін на послуги	7		+					
5	Рівень концентрації послуг	14		+					

Таблиця 4.12 - SWOT-аналіз стартап-проекту

Сильні сторони: Простота роботи з розробленою системою, зрозумілий інтерфейс,	**Слабкі сторони:** Потрібно напрацьовувати репутацію, маловідомість. Великі маркетингові затрати.
Можливості: Збільшення користувачів платформи MacOS, а отже зростання ринку. Збільшення інтересу в напрямку безпеки програмних застосунків	**Загрози:** Використання користувачами безкоштовних аналогів, піратських версій. Недовіра до репутації виробника.

Виходячи зі SWOT-аналізу стартап-проекту проаналізовано сильні та слабкі сторони проекту. Враховано подальші можливості для розвитку проекту, а також визначено основні загрози проекту. Ці данні дають можливість ще краще оцінити актуальність проблеми, та вказують на те що проект заслуговує на реалізацію.

Таблиця 4.13 - Альтернативи ринкового впровадження стартап-проекту

№ п/п	*Альтернатива (орієнтовний комплекс заходів) ринкової поведінки*	*Ймовірність отримання ресурсів*	*Строки реалізації*
1	Концентрація на одному чи двох типах клієнтів	Висока ймовірність отримання ресурсів	1 рік
2	Побудова рішення, використовуючи компоненти однієї компанії-постачальника	Висока ймовірність отримання ресурсів та залучення підтримкою вендора	6 місяців
3	Побудова гібридного сервісу	Мала ймовірність отримання ресурсів	3 роки

З означених альтернатив ринкового впровадження даного стартап-проекту було вирішено обрати побудову рішення з використанням компонентів

однієї компанії постачальника, при цьому строки реалізації становитимуть приблизно 6 місяців.

4.4 Розроблення ринкової стратегії проекту

Таблиця 4.14 - Вибір цільових груп потенційних споживачів

1	*2*	*3*	*4*	*5*	*5*
№ п/п	*Опис профілю цільової групи потенційних клієнтів*	*Готовність споживачів сприйняти продукт*	*Орієнтовний попит в межах цільової групи (сегменту)*	*Інтенсивність конкуренції в сегменті*	*Простота входу у сегмент*
1	Державні корпорації та органи державної влади	Клієнти потребують продукт такого типу та готові ним користуватись	Високий попит, пов'язаний із необхідністю послуг у сфері кібербезпеки	Невисока конкуренція в сегменті, яка пов'язана із вимогами щодо стандартизації та сертифікації	Є складність входу, яка пов'язана із негативним ставленням до хмарних сервісів
2	Провайдери	Клієнти зацікавлені продуктом	Середній рівень попиту	Середній рівень конкуренції	Є складність, яка пов'язана із кількістю та рівнем послуг, які потребують клієнти

Продовження таблиці 4.14

1	2	3	4	5	6
3	Компанії середнього і великого бізнесу (medium and large enterprises - MLE) в Україні і за кордоном	Клієнти потребують продукт такого типу та готові ним користуватись	Високий рівень попиту, пов'язаний із необхідністю послуг у сфері кібербезпеки	Дуже високий рівень конкуренції	Є складність, пов'язана з ціновою конкуренцією на послуги

4	Поодинокі користувачі (фізичні особи)	Споживачі зацікавлені продуктом то готові ним користуватись	низький попит, пов'язаний із високою вартістю послуг	Практично відсутня конкуренція	Легкий вхід, але низька рентабельність проекту
Які цільові групи обрано: державні корпорації та органи державної влади, провайдери ISP та компанії середнього і великого бізнесу.					

Проаналізувавши ринкову стратегію було прийнято рішення використовувати стратегію диференційованого маркетингу.

Таблиця 4.15 - Визначення базової стратегії розвитку

№ п/п	*Обрана альтернатива розвитку проекту*	*Стратегія охоплення ринку*	*Ключові конкурентоспроможні позиції відповідно до обраної альтернативи*	*Базова стратегія розвитку*
1	Флангова атака	Стратегія диференційованого маркетингу	Сильні сторони та можливості рішення	Стратегія диференціації

Таблиця 4.16 - Визначення базової стратегії конкурентної поведінки

№ п/п	*Чи є проект «першопрохідцем» на ринку?*	*Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?*	*Чи буде компанія копіювати основні характеристики товару конкурента, і які?*	*Стратегія конкурентної поведінки*
1	Ні	Так	Так. Використання SIEM комплексу, консолі керування тощо	Стратегія виклику лідера

Таблиця 4.17 - Визначення стратегії позиціонування

№ п/п	*Вимоги до товару цільової аудиторії*	*Базова стратегія розвитку*	*Ключові конкурентоспроможні позиції власного стартап-проекту*	*Вибір асоціацій, які мають сформувати комплексну позицію власного проекту*
1	Отримання комплексних послуг з «одних» рук. Зниження видатків з бюджету на забезпечення захисту. Отримання знань від спеціалістів щодо інформаційної безпеки. Отримання єдиного інструмента керування безпекою та комплексного бачення стану захищеності. Мінімізація фінансових та репутаційних втрат, забезпечення інформаційної безпеки як складової безпеки бізнесу. Забезпечення захищеності інформаційної структури. Виконання вимог регулятора	Стратегія диференціації	Сильні сторони та можливості рішення	Моніторинг безпеки інформаційних систем клієнтів. Розслідування інцидентів, аудит консалтинг в сфері інформаційної безпеки.

5. Розроблення маркетингової програми стартап-проекту

Таблиця 4.18 - Визначення ключових переваг концепції потенційного товару

1	*2*	*3*	*4*
№ п/п	*Потреба*	*Вигода, яку пропонує товар*	*Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити*
1	Моніторинг безпеки інформаційних систем підприємства чи організації	Забезпечення захищеності інформаційної структури	Отримання комплексних послуг з «одних» рук. Зниження видатків з бюджету на забезпечення захисту. Отримання знань від спеціалістів щодо інформаційної безпеки

2	Розслідування інцидентів, аудит консалтинг в сфері інформаційної безпеки	Високий рівень відмовостійкості та безперервності роботи рішення і надання послуг	Кваліфіковані та сертифіковані кадри для виконання поставлених задач (розслідування інцидентів, консультації, проведення аудитів, аналітика тощо)
3	Отримання єдиної консолі керування безпекою та аналітики	Консоль керування безпекою	Надання широкого спектру послуг, що не має аналогів в Україні; гнучкість цін на послуги

Таблиця 4.19 - Опис трьох рівнів моделі товару

Рівні товару	*Сутність та складові*		
I. Товар за задумом	Моніторинг безпеки інформаційних систем клієнтів. Розробка та розгортання «хмарної» консолі керування безпекою. Можливість розслідування інцидентів, проведення аудитів та консалтинг в сфері інформаційної безпеки.		
II. Товар у реальному виконанні	Властивості/характеристики	М/Нм	Вр/Тх /Тл/Е/Ор
	1. Використання хмарних сервісів для розміщення рішення	М	Вр/Тх/Тл
	2. Використання комплексу SIEM Splunk	М	Тх/Тл/Е
	3. Використання платформ, що надають інформацію про загрози, індикатори компрометації, оновлення та виправлення для програмного забезпечення, загальний стан інформаційної безпеки	Нм	Вр/Тл/Е
	4. Можливість налаштування консолі програмного комплексу рішення під будь-які потреби користувача	М	Тх/Е/Ор
	5. Швидкість роботи рішення для користувача	Нм	Тл/Е/Ор
	6. Надання доступу до «хмарної» консолі з будь-якої точки світу	М	Тл/Е
	Якість: Забезпечення захищеності інформаційної структури. Виконання вимог регулятора		

	Надання користувачу електронної ліцензії (ключа доступу) до хмарного сервісу
	Марка: Objecteve-C
III. Товар із підкріпленням	До продажу: для стимулювання попиту на продукт можна розробити програму надання тимчасового доступу до рішення та отримання послуг в обмеженому обсязі (тестування продукту).
	Після продажу: Розробка та проведення рекламної кампанії
За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності	

Таблиця 4.20 - Визначення меж встановлення ціни

№ п/п	*Рівень цін на товари-замінники*	*Рівень цін на товари-аналоги*	*Рівень доходів цільової групи споживачів*	*Верхня та нижня межі встановлення ціни на товар/послугу*
1	20000 ум.од	650 ум.од	Високий або середній	Ціна підписки на сервіс моніторингу/керування безпекою – 350-700 ум.од.; ціна послуг консалтингу – 100-300 ум.од.

Таблиця 4.21 - Формування системи збуту

№ п/п	*Специфіка закупівельної поведінки цільових клієнтів*	*Функції збуту, які має виконувати постачальник товару*	*Глибина каналу збуту*	*Оптимальна система збуту*

1	прийняття рішення про необхідність систем моніторингу та керування безпеки, вибір джерела задоволення своїх потреб для забезпечення	пристосування збутової мережі до запитів споживачів; пошук перспективних засобів просування товарів; розробка та вдосконалення маркетингової політики; вибір посередників	Дворівневий канал збуту (із залученням посередника та дистриб'ютора)	Залучення компаній посередників та партнерів для формування системи збуту

Таблиця 4.22 - Концепція маркетингових комунікацій

№ п/п	*Специфіка поведінки цільових клієнтів*	*Канали комунікацій, якими користуються цільові клієнти*	*Ключові позиції, обрані для позиціонування*	*Завдання рекламного повідомлення*	*Концепція рекламного звернення*
1	Дослідження властивостей та якостей рішення, можливість тестування продукту, прийняття рішення про необхідність використання продукту для задоволення своїх потреб	Канали інтегрованих маркетингових комунікацій	Використання сильних сторін рішення: хмарний сервіс; надання широкого спектру послуг, що не має аналогів в Україні; незалежність від цін за обслуговування серверних потужностей; швидке впровадження нових технологій	повідомлення, пов'язані з особистою вигодою аудиторії, що показують, як товар може задовольняти потреби покупця;	реклама, що демонструє якість товару, його економічність, цінність або можливості експлуатації

Висновки до розділу 4

Проаналізовані данні та розроблений стартап-проект показує непогані перспективи та можливості росту при теперішньому стані ринку та конкурентів.

Запропоноване рішення буде мати гарний попит серед різних груп користувачів. Проблема є актуальною тому, що з кожним роком кількість користувачів збільшується, а це означає що платформа MacOS стає все більш привабливою для атак. Тому системи захисту будуть користуватися попитом.

Проект має непогану рентабельність на ринку послуг такого роду. Також є і негативні сторони у вигляді труднощів пов'язаних з конкурентами. Необхідність впровадження складної маркетингової компанії для розвитку бренду продукту. Аналіз стартап-проету показує, актуальність та можливість реалізації даного проекту.

ВИСНОВКИ

В ході нашого дослідження було розглянуто систему перевірки сертифікації застосунків під операційною системою MacOS. Знайдено та проаналізовано виконувані файли операційної системи MacOS. Розглянути підсистеми перевірки сертифікатів. Проаналізовано переваги та недоліки цих систем. Розроблено програмне рішення однієї з вразливостей системи перевірки підпису Codesign. Використано ці методи на реальному проекті.

Виявлено схему та послідовність взаємодій, які використовуються під час сертифікації застосунків MacOS; Проаналізовано будову Mach-O Fat бінарних файлів. Вивчено наявну вразливість системи перевірки сертифікатів Fat бінарних файлів, Запропоновано методику перевірки наявності вразливості для довільного застосунка MacOS. Розроблено програмне рішення, яке втілює основні положення запропонованої методики.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1 Інфраструктура відкритих ключів [Електронний ресурс] - https://uk.wikipedia.org/wiki/Інфраструктура_відкритих_ключів

2 Компанія Apple Inc [Електронний ресурс] - https://uk.wikipedia.org/wiki/Apple_Inc.

3 Загальні відомості про World Wide Developers Conference [Електронний ресурс] - https://uk.wikipedia.org/wiki/Worldwide_Developers_Conference

[Електронний ресурс] - https://uk.wikipedia.org/wiki/Apple_ID

4 Принципи перевірки підпису коду на Windows [Електронний ресурс] - https://habr.com/company/tuthost/blog/152867/ Codesign on Windows

5 Принципи роботи системи Code signing [Електронний ресурс] - https://en.wikipedia.org/wiki/Code_signing

6 Code Sign Integrity [Електронний ресурс] - https://developer.apple.com/library/archive/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html

7 Apple application certificates [Електронний ресурс] - https://developer.apple.com/support/certificates/

8 Deep meaning and understanding of CodeSign [Електронний ресурс] - https://help.apple.com/xcode/mac/current/#/dev3a05256b8

9 Сертификация в Apple Developer Center простым и понятным языком [Електронний ресурс] - https://habr.com/post/280626/

10 Mach-O Programming Topics [Електронний ресурс] - https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/

MachOTopics/0-Introduction/introduction.html#//apple_ref/doc/uid/TP40001827-SW1

11 Official header of machine.h file [Електронний ресурс] - https://opensource.apple.com/source/dtrace/dtrace-78/head/arch.h

12 The 32-bit mach header appears at the very beginning of the object file [Електронний ресурс] - https://opensource.apple.com/source/xnu/xnu-1456.1.26/EXTERNAL_HEADERS/mach-o/loader.h

13 PARSING MACH-O FILES [Електронний ресурс] - https://lowlevelbits.org/parsing-mach-o-files/

14 Mach-O-View software [Електронний ресурс] - https://sourceforge.net/projects/machoview/

15 Hopper application in details [Електронний ресурс] - https://www.hopperapp.com

16 Mac-O Segment Dumper [Електронний ресурс] - https://github.com/AlexDenisov/segment_dumper

ДОДАТОК

Лістинг програми-парсера для бінарних файлів Mach-O:

```
//  main.c
//  MachOHeaderParser
//
//  Created by Рома on 12/26/16.
//  Copyright © 2016 RomanTikhonychev. All rights reserved.

#include <stdio.h>
#include <stdlib.h>
#include <mach-o/loader.h>
#include <mach-o/swap.h>

void dump_segments(FILE *obj_file);

int main(int argc, char *argv[]) {
    //const char *filename = argv[1];
    FILE *obj_file = fopen("ButtonBinding2", "rb");
    dump_segments(obj_file);
    fclose(obj_file);
    return 0;
}

uint32_t read_magic(FILE *obj_file, int offset) {
    uint32_t magic;
    fseek(obj_file, offset, SEEK_SET);
    fread(&magic, sizeof(uint32_t), 1, obj_file);
    return magic;
}

int should_swap_bytes(uint32_t magic) {
    return magic == MH_CIGAM || magic == MH_CIGAM_64;
}

int is_magic_64(uint32_t magic) {
    return magic == MH_MAGIC_64 || magic == MH_CIGAM_64;
}

void *load_bytes(FILE *obj_file, int offset, int size) {
    void *buf = calloc(1, size);
    fseek(obj_file, offset, SEEK_SET);
    fread(buf, size, 1, obj_file);
    return buf;
}

void dump_mach_header(FILE *obj_file, int offset, int is_64, int is_swap) {
    if (is_64) {
        int header_size = sizeof(struct mach_header_64);
```

```
        struct mach_header_64 *header = load_bytes(obj_file, offset, header_size);
        if (is_swap) {
            swap_mach_header_64(header, 0);
        }

        free(header);
    } else {
        int header_size = sizeof(struct mach_header);
        struct mach_header *header = load_bytes(obj_file, offset, header_size);
        if (is_swap) {
            swap_mach_header(header, 0);
        }
        free(header);
    }
}

void dump_segments(FILE *obj_file) {
    uint32_t magic = read_magic(obj_file, 0);
    int is_64 = is_magic_64(magic);
    int is_swap = should_swap_bytes(magic);
    dump_mach_header(obj_file, 0, is_64, is_swap);
}
```